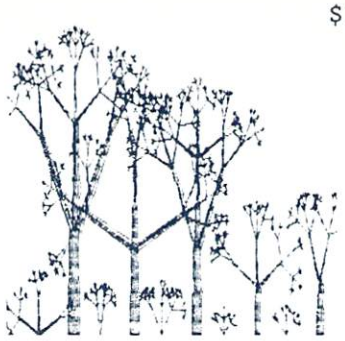
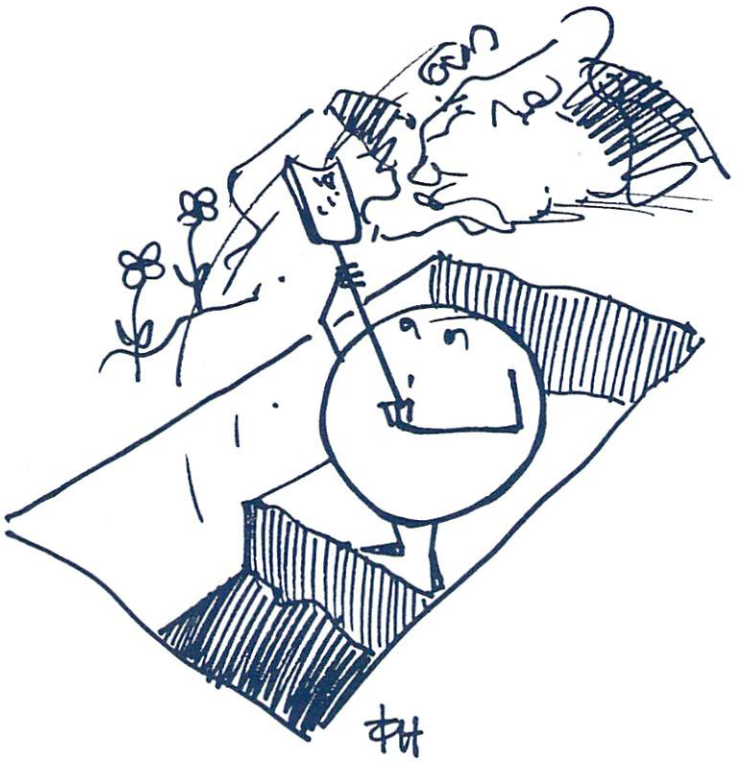
COMAL TODAY 8

$3.95



RANDOM FOREST - page 26



WORD GAME

DIRECTORY EDITOR

RANDOM FOREST

EASY SPRITES

SHAPE MAKER

CUSTOM ERROR MESSAGE

MUSIC WRITER

SOUNDEX

INTERRUPT COMMAND

FUN PRINTING

COMAL TODAY
COMAL USERS GROUP, U.S.A., LIMITED
6041 Monona Drive, Madison, WI 53716

BULK RATE
U.S.POSTAGE
PAID
MADISON, WI
PERMIT 2981

IF YOUR LABEL SAYS
LAST ISSUE: 8
YOU MUST RENEW NOW.
USE CARD INSIDE

COMAL TODAY ISSUE 8 July 17, 1985

EDITOR

Len Lindsay

ASSISTANTS

Debra R Junk
David Stidolph
Maria Lindsay
Denise Bernstein

PHOTOS / ART

Ray Hansen
Frank Hejndorf

CONTRIBUTORS

Phyrne Bacon
Denise Bernstein
Dutch COMAL Group
John Eldredge
Michael Erskine
Kendall French
Daniel Horowitz
Debra Junk
Dick Klingens
Jesse Knight
Len Lindsay
Phred Linn
Ian MacPhedran
Larry Phillips
Kevin Quiggle
Robert Ross
Gordon Shigley
Robert Shingledecker
David Stidolph
Ken Strahl
David Tamkin
UniComal ApS
Larry Winckles
Chris Zamara

PUBLISHER

COMAL Users Group, USA, Limited
6041 Monona Drive
Madison, WI 53716
608-222-4432

From the Editor's Disk

Exciting times. Each month COMAL seems to climb higher. We have COMAL 2.0 in stock for the IBM PC and compatibles. Write programs for the IBM PC on your C64. And the C64 COMAL 2.0 Deluxe Cartridge Pak is now only $84.95 to subscribers, complete with 2 books and 5 disks.

QUICKLY NOW

We've resolved a common complaint about C64 COMAL 0.14 - the 90 seconds loading time. COMAL QUICK 0.14 works with both the MSD and 1541 (this may be the FIRST fast loader for the MSD). And the price? Only $10.95 to subscribers. And it includes UTILITY DISK #2 with instruction book. Get your COMAL QUICK now (see page 80).

QUICK CHIP is also available for the empty socket of the COMAL 2.0 Cartridge. It allows COMAL 2.0 programs to automatically LOAD several times faster than normal. Call for price and ordering instructions.

LINCOLN COLLEGE COMPUTER CAMP SUCCESS

COMAL did well at camp. In our spare time during our 2 hour COMAL class for the 5 days, we wrote and tested a small data base to keep track of people. Using modules (one of COMAL's specialties) different people wrote various parts of the program. Then we merged it together. I had fun demonstrating it to all the camp attendees. By the end of the camp I estimate that 2/3rds of the attendees had a COMAL 2.0 Cartridge.

THE 2 HOUR SPECIAL PROJECT

A few weeks ago I volunteered to help someone transfer a small data base of information from a C64 to an IBM. He brought over an IBM disk with the information on it. All I had to do was reformat the data so PC FILE would accept it. I spent two hours trying to figure out what PC FILE wanted. Finally, I gave up and declared that I could write an entire system to manage the data in COMAL in that time. Two hours later I had his system done AND tested! IBM PC COMAL saved the day.

MARCA COMPUTER SHOW

COMAL will be a highlight of the biggest Commodore Show in North America. Three different COMAL sessions are scheduled, including one with the Father of COMAL, Borge Christensen, direct from Denmark. COMAL QUICK and the QUICK CHIP will be shown for the first time. And our new Royal COMAL shirt will make a splash. We plan to have a couple hundred people wearing the shirts at the show. Watch out for the COMAL controlled LEGO ROBOT patrolling our booth.

A NEW ERA BEGINS

The days of listing LONG programs in magazines for readers to TYPE IN are over. Long program listings are error prone. Either the reader will make an error typing in the program, or the listing will be printed incorrectly. The longer the program listing, the greater the chance for error. But program listings are helpful to show a program's structure. We will list programs for your information, but supply the programs, ready to run, on disk. This means an added expense, but we can do it. One of our specials this issue is a 6 disk subscription for only $29.95. This is the lowest disk subscription price we are aware of. And remember, our disks contain programs on both sides now. That comes to $2.50 per side full of programs.

HOW TO TYPE IN COMAL PROGRAMS

Line numbers are irrelevant to a running COMAL program. COMAL only provides line numbers for our benefit in editing the program. Thus most magazines do not use line numbers when listing a COMAL program. It is up to **YOU** to provide the line numbers. But of course, **COMAL can do it for you** quite easily. Just follow these steps to type in a COMAL program listing:

1) Enter command: **NEW**
2) Enter command: **AUTO**
3) Type in the program
4) When done:
   Version 0.14: Hit <RETURN> key twice
   Version 2.0 : Hit <STOP> key

Remember - use unshifted letters thoughout entering the program. If letters are capitalized in the listing it does not mean use the SHIFT with those letters. They are capitalized merely for a pleasant, easy to read, look. The only place to use SHIFTED letters is inside quotes. Also, you don't have to type leading spaces in a line. They are listed only to emphasize structures. You **DO** have to type a space between COMAL words in the program.

LONG PROGRAM LINES: We are continuing to print COMAL TODAY with two columns per page, printed in elite (12 pitch) with 40 characters maximum per line. This makes it easiest to read. However, some program listings have program lines that extend beyond the 40 character limit. We decided to list these lines in the same manner that COMAL uses when listing long lines on a 40 column screen. We simply break the line right at the 40 character spot, and continue it on the next line, indenting it properly to keep the program structures obvious. These are called wrap lines. To draw your attention to these continued lines we add a ***//wrap line*** comment to the end of the line. Whenever you see this make sure you type both lines as one continuous program line! The following example includes a line with more than 40 characters that we must list on two lines, but you must type in as one long program line:

```
IF CURRENT'NAME$<>"STOP" THEN PRINT'LABE
L(CURRENT'NAME$,PHONE$) //wrap line
```

If you type in this long program line as two shorter program lines, COMAL will not object (although sometimes it will)! But, the program will not work unless it is entered as one long line. The procedure name PRINT'LABEL is split onto two lines in the listing, but the //wrap line draws your attention to this fact.

Letters

Friends:

COMAL 2.0 has changed me from a boot and go computer slave into a creative computer user, and I have a nightmare magazine type-in program to thank for this transformation. Like most non-programming computerists I could not pass up what appeared to be a useful published program that could be mine simply for the mechanical act of typing it in.

The program was an outline editor and covered four pages with very small type. I sat down in ignorance, worked my way through exhaustion, and stood up in complete frustration. Strange messages were appearing all over the screen: type mismatch, out of data, syntax error, among others. Checking every letter and symbol over and over convinced me I had entered everything correctly. Why wouldn't it run?

Right then I started a new project: I set out to learn BASIC. Eighty dollars worth of books and manuals later, I became convinced I was part of some elaborate joke. BASIC was like a wind-up toy that once set in motion might eventually reach its goal, but would waste eighty percent of its time wandering around in circles. This couldn't be the ultimate answer. I started collecting data on other high level languages.

COMAL 0.14 was available in my users group library and the disk became buried under a mountain of other material. Every language had at least one strong feature, but no language provided a comfortable total enviornment. Then I booted COMAL.

The first thing I noticed was the clarity of the logic in program listings. Next I noticed familiar key words from BASIC in a context that made sense. I had a new project: COMAL.

The learning to program project has turned into the learning to program adventure. Instead of legions of BASIC complainers, I have fallen in with a dedicated group of COMAL explorers. My problem solving abilities on all levels have profited from my exposure to COMAL. A secondary benefit of this experience has been a healthy suspicion of the integrity of any set of high computer priests that would force any version of BASIC, as a standard, on the novice user.

Thank you. --D.R. BREA, CA

[Ed. Note: We find that the longer the program listing, the more chance of errors - either in typing it in, or in the actual listing printed. We feel that the age of typing in programs is over. We list programs for your information, so you can see how they work. Then we publish them on disk so there is no need to have to type them in.]

Editor:

I made an odd discovery. I was running COMAL 0.14 with my COMAL 2.0 cartridge plugged in and found that the space bar will pause a listing of a COMAL 0.14 program. --GM, Orangeville, Ont.

[Ed. note: It always could! The cartridge is not needed. SPACE BAR pause of program listing is a built in feature of both COMAL 0.14 and 2.0.]

Greetings:

I had an unbelievable catastrophy happen two weeks ago. My COMAL 2.0 Cartridge was stolen! It happened in St. Paul. My luggage was stolen out of a locked vehicle in front of the Holiday Inn. The luggage had not only my clothes, but my complete collection of COMAL manuals, demo disks and my cartridge. I had it with me to demonstrate a program I developed to use for an advertising display for retail stores.

After I got over the shock and filed my insurance claims, I started to write to you at once to order another cartridge and manual. But then I read about the C128. It occurs to me that I may have more than one option with this machine, since I could use the cartridge in the C64 mode, or a disk based COMAL in the CP/M mode.

Can you compare the versions available? Thank you for your advice. Any sympathy you could extend would also be warmly received. --DJ, Wadena, MN

[Ed. note: The C64 COMAL 2.0 Cartridge is the best way to go with the C128. It works in the C64 mode. The CP/M COMAL costs over $200, has no USA distributor and is not nearly as good.]

Dear Editor:

I would like to know if you're preparing materials to teach the fundamentals of COMAL 2.0. I do not know BASIC so I won't have to unlearn any bad habits. I do not want to learn COMAL 0.14 and then move up to 2.0. I hope you will be able to support the 2.0 language. We have some people in our computer club interested in it because of its greater speed and extra functions. --HC,Parsonburg, MD

[Ed. note: The cartridge tutorial binder with disk gives all the basics of using COMAL 2.0. Also, our newsletter will continue to support both 0.14 and 2.0 COMAL.]

Dear Editor:

In Today #7, Jim White writes of using Compute! Magazine's Turbodisk Program with COMAL 0.14. He's right -- it loads COMAL in about 15 seconds and certainly is cheap enough. A warning is in order though. I first used Turbodisk with an early COMAL disk with no problems, but a later disk with the improved RAM errors routine and expanded RAM procedure in the "Hi" program crashed as soon as an error occurred while typing a program. A SYS 49155, to disable Turbodisk, inserted at the beginning of the "Hi" program solves the problem nicely.

A quibble about the boot program in Jim's article -- there's no need to run the DOS wedge to load and run Turbo disk. Why waste the time when a simple routine like:

```
5 if a=0 then a=1:load"turbodisk.obj,8,1
10 sys 49152
```

at the beginning of his program would suffice. The routine could also be added to the "BOOTCOMAL" programs on most COMAL disks.

Finally, the original Turbodisk program published in Compute! requires that your printer interface be disconnected from the cassette port and the printer be turned off. Otherwise, the drive runs on and on without loading anything. However, an updated version of Turbodisk published in Compute!'s Gazette, July 1985, does not have this requirement and still works fine with COMAL 0.14. I'd advise your readers to use this version. --LA, Iron Station, NC

[Ed.note: Better yet, get our very own COMAL QUICK - a quick loading COMAL that works on both MSD and 1541 disk drives.]

Editor:

I have the 2.0 Cartridge and have found it to be even better than expected. The disk version was only a hint of COMAL's power and ease of use in programming. I have not had a single case of computer crash or lock-up with COMAL. Keep up the good work! If IBM adds some advertising power, the COMAL message will become louder and should spread faster. --JF, Mission Viejo, CA

Dear Captain COMAL:

You guys/gals are doing a great job of spreading COMAL! Providing inexpensive disks is the best way to spread any new language.

I suggest you try to include one good quality game in COMAL on each Today Disk. It would be interesting to compare public domain games in BASIC and COMAL. My ten and a half year old daughter thinks computers are boring, and the COMAL programs I've seen don't help much! COMAL man does not live by utilities and demos alone! --SC,Novato, CA

[Ed. note: OK, we'll include the games if readers submit the programs. We do have a word game in this issue. Unfortunately, most games will be long programs and may not be listed in the newsletter, but only included ready to run on the disk. Luckily we now have the lowest disk subscription price we know of in the industry.]

Friends:

I have never been more pleased with my C64. It's like a new machine. Unfortunately, I can't bear to do anything in BASIC anymore. Ah, well. Many thanks. --D.H. Northampton, MA

Your Orders

Usually it takes only a few days after we receive your order to process it, but please allow 2-4 weeks. When ordering from Canada or overseas, make sure the payment is in US Dollars (VISA/MasterCard is OK). COMAL TODAY newsletters are sent FIRST CLASS to Canada rather than bulk rate. That is why there is a $5 extra charge for a Canadian subscription.

Any changes needed in your subscription address? Just send us a postcard. Make sure to include your Subscriber Number.

Due to problems, we have discontinued the 800 order line. Please use our regular phone number for orders now. You will get much faster service and we can help you choose between various options. This fall we hope to expand the hours into the evening, but for now the hours are 8:00 am until 4:30 pm.

ALWAYS include your Subscriber Number. Starting August 1, 1985, orders without a subscriber number will not receive the special subscriber discounts, since the computer is now doing that automatically based on the subscribers number. And make sure we have your street address since we use UPS to deliver our packages.

We only can store one address per subscriber. Whenever you order something, we update the address in the file with the address you use for the order. This way we make sure your address is current. But shipping to an address different than your subscriber address requires specially handling.

Technical questions can be answered by phone Monday, Wednesday, and Friday from 11:30 am until 3:00 pm (Central Time). Try to limit your calls to 10 minutes since many others may be trying to call in. Or send in your questions along with a Self Addressed Stamped Envelope (SASE) for a written reply.

Questions and Answers

QUESTION:

I'm experiencing difficulties with the last two TODAY DISKS. When I tried to load #5 I got ?File Not Found Error. When I tried to load the COMAL 0.14 side of #6, I got Syntax Error in 9216. The COMAL 2.0 side ran fine. --JS, Minnesota

ANSWER:

The problem you are having is understandable, since we are no longer putting the BASIC boot program and COMAL on every TODAY DISK. We stopped putting these on our disks because they take up approximately 20% of the disk space --- which means fewer programs. You should load COMAL from another disk. Once COMAL is running, remove that disk and insert the new disk in the drive and type:

CHAIN"HI"<RETURN>

Once COMAL is loaded you may remove and insert any disk as needed.

QUESTION:

I recently bought a COMAL 2.0 Cartridge and disks 1,2,3,and 4 to go with it. On disk #3 is a program called "check cartridge." In using this test on my cartridge, it indicates that overlays 0,1, and 2 are bad. Do I have a bad cartridge or a bad disk?

ANSWER:

Your cartridge is fine. The check program was written for the gray EPROM cartridges. You have a black ROM cartridge. An updated version of the check program is listed in COMAL Today #6 and is on Today Disk #6, #7 and #8. Sorry for the confusion.

QUESTION:

What is the FRAME command used for? According to my COMAL pocket card, it is used to "set up a screen window," but when I have tried to use it in my programs, in either text mode or graphics mode, it doesn't seem to do anything. Can the hi-res and turtle graphics be used in the bit-mapped multi-color mode? --J.E. Tennessee

ANSWER:

FRAME cannot be used in text mode. It is used only if you wish to restrict the drawing area of the turtle. The turtle can only draw inside the frame; it can move outside the FRAME, but then it won't leave a line. Turtle graphics works in both mode 0 (HiRes) and 1 (multicolor).

QUESTION:

There is a bug in COMAL 2.0:

```
use system
quote'mode(false)
10a$=""0""+"2"
quote'mode(true)
```

Now you can edit, but not list the line! --DP, Utah

ANSWER:

The problem you are having with quote mode is not with COMAL, but with the screen editor. Try ""13"" instead of ""0"" and you'll see what I mean. The screen editor has no symbols for either of these like it does for Home or Clear. That is why the special double quotes for control numbers notation that COMAL 2.0 uses is very nice at times.

QUESTIONS:

Last month I ordered and received a copy of COMAL TODAY #5. I also was fortunate to find someone who had a copy of COMAL 0.14 and the COMAL HANDBOOK. I tried some of the programs in the book and was intrigued. I have several questions which I am hoping you will answer for me.

1. Which book should I order first as far as learning to program in COMAL?

2. Why is it when I typed in the sample program from page 7 of the HANDBOOK, line #3, the cursor blinks over the colon after the #1, and gives me a syntax error, but when I typed in a semi-colon the program continued, except when I try to run the program I receive an error message on line 0030 UNKNOWN VARIABLE.

3. I typed PRINT "[CLR]", and instead of the screen clearing, [CLR] printed on the screen.

4. When I wanted to go to BASIC, I typed in sys64790, and the program crashed. When I typed in capitals, I receive FORMAT ERROR.

I truly would appreciate a response to these questions. --D.R. Ronkonkoma, NY

ANSWERS:

1. We have found that most people like either FOUNDATIONS IN COMPUTER STUDIES WITH COMAL or BEGINNING COMAL as a first book for learning COMAL. The COMAL HANDBOOK is a fine reference book, but when just starting, a tutorial is best to learn with.

2. For the benefit of our readers, look on page 7 of the first edition HANDBOOK or page 29 of the second edition. Remember that the COMAL HANDBOOK is documenting both the disk loaded COMAL (version 0.12 or 0.14) and the full COMAL (1.02 or 2.0). You are using the disk loaded COMAL 0.14. Therefor you only can use the commands included with it. You were trying to run a program demonstrating the PRINT AT command of COMAL 2.0. If you are using COMAL 0.14 you must check at the top of each KEYWORD identification section. Look where it says VERSION 0.12 in the first edition or VERS 0.14 in the second edition. If there is a minus there (-) that means that the keyword is not available in the disk loaded COMAL 0.14. You must skip those pages unless you have the COMAL 2.0 Cartridge.

3. The Commodore 64 has several special keys. To tell you when to press one of them, the HANDBOOK and COMAL TODAY use a special notation. This is explained in the HANDBOOK on page xviii of the first edition and page 7 of the second edition. It says that when you see [CLR] in a program listing that you are supposed to type the CLR key. So you should type the PRINT and the quote ". Then instead of typing a square bracket etc, you should simply press the CLR key (hold the SHIFT key at the same time). On your screen you now will see a reverse field heart. Now you know why we printed [CLR] in the listing. If we had printed a reverse field heart - how would you know what key to press? Take a minute and look over the special notation used for the special Commodore keys.

4. If you want to go back to BASIC from COMAL - don't use a SYS command. There is a built in command to do it for you. Just type this: BASIC and press the RETURN key. It is explained on page 10 of the first edition HANDBOOK and page 34 of the second edition. Also, if you type in capitals (I assume you mean SHIFTED letters) COMAL does not understand. Several places in the HANDBOOK's introduction you are reminded NOT to use SHIFTED letters.

QUESTION:

One thing I can't figure out is the way COMAL uses punctuation. It seems very lax with the quotation marks and the use of the apostrophe seems to be stuck hither and yon. --S.L., Texas

ANSWER:

Your question about apostrophes is easily answered. COMAL is word-based, not line number based. COMAL looks at the words you put on each line and tries to figure out what they represent. The rules say that a procedure NAME may not have a space in it, so COMAL programmers, in an attempt at clarity, use an appostrophe (') for a space. Thus a procedure called GRAND TOTAL may be written as GRAND'TOTAL. This habit carries over into filenames, also, although it is not necessary.

COMAL is not so picky about quote marks. If the interpreter sees the word LOAD or SAVE, it knows a quote mark will surely follow. So any of these are OK:

```
LOAD"filename"
LOAD   "filename"
SAVE"filename
SAVE "filename
```

Notice the trailing quote mark is not needed. COMAL will add it for you. Most other keywords require a space between the keyword and what follows it, like this:

```
FOR X=1 TO 100 DO
```

not like BASIC:

```
FORX=1TO100
```

The forced spaces make the program much easier to read. Also, COMAL allows only one operation on each line, unlike messy BASIC. This makes program debugging much easier.

QUESTION:

I believe there is a problem with this disk. When I enter "c return" to get into COMAL, I get a "end at 0630." In the graphics mode I can change border color, but if I try to change BACKGROUND I get no response. Could you check this disk out? --J.E. Wisconsin

ANSWER:

Your disk was just fine. Type option "c" puts you into the COMAL system. The "end at" message is the Demo/Hi program ending ---giving you control of COMAL. Now YOU must do something. For example, turn on your C64 and it doesn't do anything. It is working just fine, and is only waiting for a program or command. The same with COMAL: once you choose "c", you need to load and run a program. Try: CAT. It shows you a catalog of the disk. Then LOAD and RUN any program. To change background color on the graphics screen:

```
BACKGROUND 2
CLEAR
```

You must issue the command CLEAR to affect the graphics screen. This is explained on page 78 of the book COMMODORE 64 GRAPHICS WITH COMAL. ■

COMPUTER PEOPLE-

You know:
The one's that stay up all night,
Sleep 'till noon.
The all night geniuses!

We Non-Computer people must give you credit.

You punch those keys into some WILD Morse-Code,
Save it on that Floppy flat disk
And Presto:
There's a program that can do a lot of my work for me.

I Like That!! ... Denise. ■

The Truth About Computers

by Debra Ruth Junk

Two erroneous beliefs concerning computers have taken root in American society. One holds that computers are logical, scientific and programmer-controlled. The other declares that computers are essentially stupid and do only what they're told to do. Both of these positions are obviously false, yet many innocent and unwary computerists have been lulled into believing them.

The real truth, which no computer journal has yet dared to reveal, is that computers are MAGICAL. Computer magic is both powerful and frightening, so the extent of the coverup is hard to determine.

Computer salesmen in their three-piece suits try to make computers look like just another tool in the hardware store. But listen to these instructions for a program: "Move, relative to the current position, to a new current position. This will change the current position by adding 5 to the current x position, and 9 to the current y position. There is no change to the screen." Does that sound to you like instructions on how to use a wrench or more like an incantation??

The deception is aided by the ordinary appearance of computer equipment. The screen looks like a TV. The keyboard is quite similar to a typewriter. The disks that are used for "programs" (or spells, to be more accurate) look like 45 rpm records.

Although appearances are deceiving, the minute you turn your computer on you should be able to tell it is NOT "just another tool." The first sound you hear is the beeping and bubbling of the cauldron starting to boil. Does your toaster make sounds like that when you turn it on?

Once the cauldron is simmering, you can put a disk into the computer drive, say a few magic words, and get an image to appear on the screen. Nothing too spooky here, until you take the disk out and the image won't disappear. No matter what you do to the disk the image will not get off the screen! Wouldn't you wonder what was up if the music kept on playing after you removed a record from your stereo turntable? Or if you took the slides out of a projector and the image stayed on the screen? Why hasn't anyone reported these computers to Ghostbusters?

Actually, almost all of the magic of computers rests in these disks. They are extremely sensitive critters, however, so you must be very, very careful with them if you want the magic to work. First and foremost, you should be aware that they don't like to be touched. This means that when you are not using them for a specific task, they should be kept in a paper cloak or sleeve. (The paper holds in the magic so it can't escape until needed.)

When you are ready to cast a spell, gently remove the disk from the sleeve. Hold the disk at the very edge and insert it into the "disk drive." The cauldron sits inside this so-called disk drive, and the disk must make contact with it before the magic can begin. To make good magic, the label on the disk must be facing the sky (in a horizontal disk drive), or facing left in a vertical disk drive.

What specific magic acts can these disks perform? They can draw, make music, compute income taxes (this one uses black magic), send letters to your friends, run a business or conjure up a recipe for fried bat wings.

So where does COMAL fit into this scheme of things? It is simply the most powerful magic around. ■

Illusion

This program draws an optical illusion of 3 vertical bars interweaved with 3 horizontal bars. It also demonstrates the removal of hidden lines.

```
// delete "0:illusion"
//  by Captain COMAL's Friends
// save   "0:illusion"
//
//This c64 0.14 COMAL program was
// adapted from Ralph Reinhart's
// program in Turbo Pascal
// for the IBM PC
//
dim xv#(1:11), yv#(1:11)
dim xh#(1:11), yh#(1:11)
dim xb#(1:6), yb#(1:6)
dim xe#(1:6), ye#(1:6)
//
data 90,90,110,110,90,100,120,110
data 120,120,110
data 30,160,160,30,30,20,20,30,20
data 150,160
data 65,75,275,275,265,265,275
data 265,65,65,265
data 45,35,35,55,65,45,35
data 45,45,65,65
data 90,110,120,90,110,120
data 36,36,36,45,45,45
data 111,111,111,120,120,120
data 35,45,65,35,45,65
//
block2 // main program
//
proc block2
 getdata
 //
 for j:=0 to 2 do
  for i:=2 to 11 do
   draw(xv#(i-1)+j*60,yv#(i-1),xv#(i)+j
   *60,yv#(i),1) // wrap line
   draw(xh#(i-1),yh#(i-1)-j*40,xh#(i),y
   h#(i)-j*40,1) // wrap line
  endfor i
 endfor j
 //
// draw the bars
//
for k:=0 to 1 do
 for j:=0 to 1 do
  for i:=1 to 3 do
   draw(xb#(i)+j*120,yb#(i)-k*80,xb#(i
   +3)+j*120,yb#(i+3)-k*80+1,0) //wrap
   draw(xb#(i)+60,yb#(i)-40,xb#(i+3)+6
   0,yb#(i+3)-39,0) // wrap line
   draw(xe#(i)+61,ye#(i)-j*80,xe#(i+3)
   +59,ye#(i+3)-j*80,0) // wrap line
   draw(xe#(i)+j*120+2,ye#(i)-40,xe#(i
   +3)+j*120-1,ye#(i+3)-40,0) // wrap
  endfor i
 endfor j
endfor k
//
// remove the hidden lines
//
for j:=0 to 1 do
 for i:=136-j*80 to 154-j*80 do
  draw(66,i,149,i,2)
  draw(183,i,262,i,2)
 endfor i
endfor j
for i:=96 to 114 do
 draw(66,i,88,i,2)
 draw(123,i,208,i,2)
 draw(243,i,262,i,2)
endfor i
//
// fill horizontal bars
//
for j:=0 to 1 do
 for i:=92+j*120 to 108+j*120 do
  //
  draw(i,169,i,166,3)
  draw(i,134,i,86,3)
  draw(i,54,i,41,3)
 endfor i
endfor j
for i:=152 to 168 do
 draw(i,41,i,94,3)
 draw(i,125,i,169,3)
endfor i
//
```

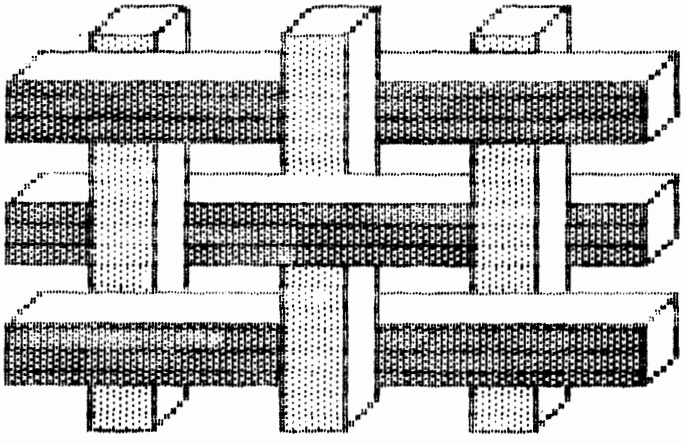
More ►

```
 // fill vertical bars
 //
 wait
endproc block2
//
proc getarray(ref a#(),1) closed
 for i:=1 to 1 do
  read a#(i)
 endfor i
endproc getarray
//
proc getdata
 restore
 getarray(xv#,11)
 getarray(yv#,11)
 getarray(xh#,11)
 getarray(yh#,11)
 for i:=1 to 11 do
  yv#(i):=200-yv#(i)
  yh#(i):=200-yh#(i)
 endfor i
 getarray(xb#,6)
 getarray(yb#,6)
 getarray(xe#,6)
 getarray(ye#,6)
 for i:=1 to 6 do
  yb#(i):=200-yb#(i)
  ye#(i):=200-ye#(i)
 endfor i
endproc getdata
//
proc wait
 while key$<>chr$(0) do null
 while key$=chr$(0) do null
 settext
 print chr$(147),chr$(14),
endproc wait
//
proc draw(a#,b#,c#,d#,e) closed
 pencolor e
 moveto a#,b#
 drawto c#,d#
endproc draw
 //
proc init'graph
 pencolor 0
 border 0
 background 0
 setgraphic 1
 hideturtle
endproc init'graph
```

2.0 Adjustments

To run this program with the 2.0 COMAL cartridge the following three procedures need to be changed.

```
PROC WAIT
 WHILE KEY$<>CHR$(0) DO NULL
 WHILE KEY$=CHR$(0) DO NULL
 textscreen
 PAGE
ENDPROC wait
//
PROC draw(a#,b#,c#,d#,e) CLOSED
 pencolor(e)
 moveto(a#,b#)
 drawto(c#,d#)
ENDPROC draw
 //
PROC init'graph
 USE graphics
 pencolor(0)
 border(0)
 background(0)
 graphicscreen(1)
 hideturtle
ENDPROC init'graph ■
```



Easy Sprites for Beginners

by Len Lindsay

Beginners take heart. You too can control the sprites in your Commodore 64. While virtually impossible from BASIC, sprite control with COMAL is quite simple, and gives you professional results. Both versions 0.14 and 2.0 have built in sprite controls, but the 2.0 Cartridge has about twice as many sprite commands built in.

What is a sprite? Glad you asked. A sprite is nothing without a shape. A sprite can assume the image of any shape, and can change images as often as it wants. It can appear on your graphics screen, just like the drawings made with turtle graphics - with one major difference: when you take the sprite away, whatever else was on the screen stays there unchanged. The sprite is like an overlay that you can move around the screen without affecting the pictures or words.

You've seen sprites in action already. Most arcade games use sprites. For example, a man running around is really a sprite. One sprite can continually change its shape, so a running man can have his legs and arms moving as he runs.

LETS DO IT - FLYING BATS

In order to show you how easy COMAL makes it, we will perform some sprite magic. All you need is your Commodore 64 computer, monitor, and disk drive - and your COMAL 0.14 disk or 2.0 Cartridge.

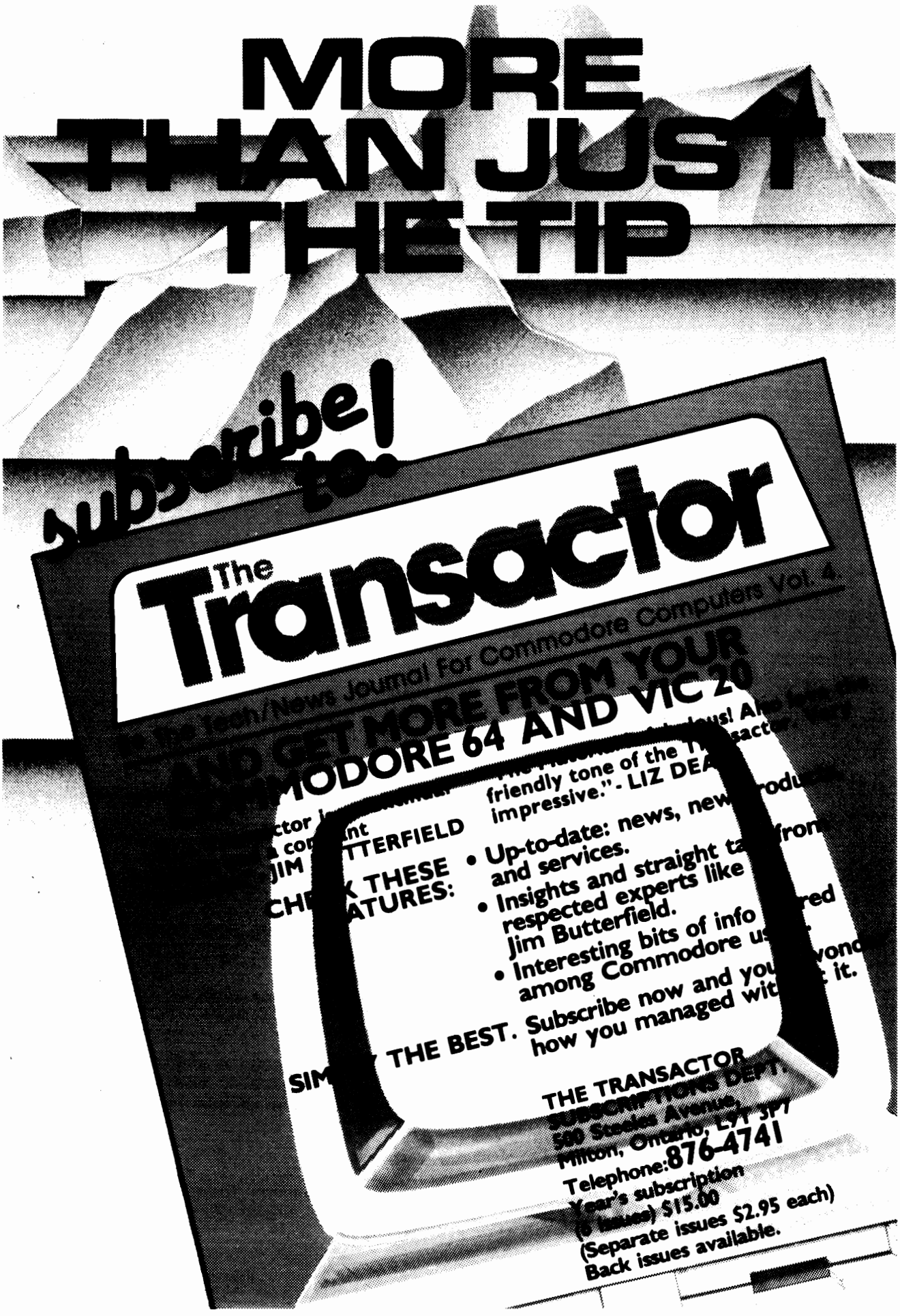
The hardest part of using sprites is creating the shapes. A good looking sprite is a work of art. Creating it takes some time and imagination. A complete sprite shape is stored as a string that is 64 characters long. The first 63 characters define the shape, the last character tells COMAL whether the shape is high resolution (2 colors only) or multi-color (4 colors, but the 'dots' are twice as wide).

The sprite definition string can be created from data statements, or can be stored on disk as a file. The advantage of a disk file image is that it can be used by any program. Create the shape once, use it many times.

You can use a sprite editor to create a sprite shape; one is included on TODAY DISK #2, BEST OF COMAL DISK and UTILITY DISK #1 (and CARTRIDGE DEMO DISK #3 for the C64 COMAL 2.0 Cartridge). However, to save you the hour needed to create the shapes, we will provide you with a short program that will create 3 different shapes that when used in sequence will look like a flying bat. Make sure you are in COMAL (either 0.14 or 2.0) - then just type in the following program (the program will also work with the COMAL 2.0 Cartridge). Save it for future use. Next, RUN the program. It will create three files on your disk. Each file contains one sprite shape. We will use these three files in the following programs.

```
print "now creating 3 shape files"
bats("shap.bat1")
bats("shap.bat2")
bats("shap.bat3")
print "all done."
//
proc bats(name$)
  open file 2,name$,write
  for x:=1 to 64 do
    read num
    print file 2: chr$(num),
  endfor x
  close file 2
  print name$;"is created."
endproc bats
```

More ►


MORE THAN JUST THE TIP
subscribe to!
The Transactor
The Tech/News Journal For Commodore Computers Vol. 4.
AND GET MORE FROM YOUR
MODORE 64 AND VIC 20
friendly tone of the
impressive." - LIZ DEA
ctor
JIM TTERFIELD
K THESE
ATURES:
• Up-to-date: news, new
and services.
• Insights and straight ta
respected experts like
Jim Butterfield.
• Interesting bits of info
among Commodore us
THE BEST. Subscribe now and you
how you managed wit
it.
SIM
THE TRANSACTOR
SUBSCRIPTIONS DEPT.
500 Steeles Avenue
Milton, Ontario, L9T 3P7
Telephone: 876-4741
Year's subscription
(6 issues) $15.00
(Separate issues $2.95 each)
Back issues available.

```
//
// bat shape number 1
data 0,0,0,0,0,0,0,32
data 64,0,246,240,3,255,252,6
data 255,246,8,95,161,0,6,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
//
// bat shape number 2
data 0,0,0,0,0,0,0,32
data 64,0,102,96,0,111,96,0
data 255,240,1,63,200,0,6,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
//
// bat shape number 3
data 0,0,0,0,0,0,0,16
data 128,0,54,192,0,63,192,0
data 95,160,0,22,128,0,6,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
data 0,0,0,0,0,0,0,0
```

Now, how do you get the shapes from the disk files into your program. With the COMAL 2.0 Cartridge it is quite simple: just use LOADSHAPE, part of the SPRITES package:

```
USE sprites
loadshape(1,"shap.bat1")
loadshape(2,"shap.bat2")
loadshape(3,"sahp.bat3")
```

These four commands can be issued from direct mode, or from within a running program. They set up three sprite definitions, numbered 1, 2, and 3.

You can do the same thing in COMAL 0.14. But LOADSHAPE is not a built in command, so we will just add it ourselves. Type in the following procedure that will add the word LOADSHAPE to COMAL 0.14 for you:

```
NEW
AUTO 9000 // start numbers at 9000

proc loadshape(spr,filename$) closed
 dim ds$ of 2
 open file 81,filename$,read
 ds$:=status$
 if ds$="00" and spr>=0 and spr<64 then
  a:=828
  poke a+0,162 //    ldx #81
  poke a+1,81
  poke a+2,32 //     jsr $ffc6
  poke a+3,198
  poke a+4,255
  poke a+5,160 //    ldy #00
  poke a+6,0
  poke a+7,32 // lp  jsr $ffcf
  poke a+8,207
  poke a+9,255
  poke a+10,153 //   sta spr,y
  poke a+11,(49152+spr*64) mod 256
  poke a+12,(49152+spr*64) div 256
  poke a+13,200 //   iny
  poke a+14,192 //   cpy #64
  poke a+15,64
  poke a+16,208 //   bne lp
  poke a+17,245
  poke a+18,96 //    rts
  sys a
 endif
 close file 81
endproc loadshape
```

Once you have typed in the procedure store it on disk for use by any future program. Store it like this:

```
LIST "LOADSHAPE.PROC"
```

Merge it with any other program later like this:

```
RENUM // get line numbers to beginning
ENTER "LOADSHAPE.PROC"
```

The RENUM command is used to make sure that no lines are in the 9000's, or they will conflict with the lines in our LOADSHAPE procedure.

Now, from COMAL 0.14, with the LOADSHAPE procedure you can issue these commands:

More ►

```
run //after RUN command LOADSHAPE works
loadshape(1,"shap.bat1")
loadshape(2,"shap.bat2")
loadshape(3,"sahp.bat3")
```

Yes, the commands are exactly the same as with COMAL 2.0. The only hitch is that you must RUN your program before COMAL knows about the new word LOADSHAPE that we have added.

COMAL 0.14 and 2.0 will share many of the commands for our sprite programs. Together we move onward.

Let's review. We now have 3 sprite shapes defined and stored on disk. We also know how to get those shapes back into our program. Now, let's see those shapes.

Wait. A shape is nothing without a sprite. We need a sprite to assume the image of the shapes we have created. There are 8 sprites available from COMAL. They are numbered, 0 through 7 (computer people like to count from 0 instead of 1). Let's use sprite number 1 for our example that follows.

OK. First we tell sprite number 1 to use the shapes we have stored with definitions 1, 2, and 3. No PEEK or POKE is necessary to do this. COMAL has built in words to take care of manipulating our sprites. All we need is a few easy to understand commands - all from direct mode too!

First let's switch to the graphics screen. That's where the action will be:

```
From COMAL 0.14:  SETGRAPHIC 0
From COMAL 2.0 :  USE GRAPHICS
                  SPLITSCREEN
```

Now we are in splitscreen mode. That means that we are using the graphics screen with the top few lines as a window into the text screen. In version 0.14 you have 2 text lines, while with the cartridge you have 4 text lines. In both versions we now can issue a few sprite commands. First let's give our sprite a position on the screen. Coordinates 50,50 should be OK to start with:

```
COMAL 0.14:  SPRITEPOS 1,50,50
COMAL 2.0 :  SPRITEPOS(1,50,50)
```

That gave our sprite number 1 a position of 50,50. If you were lucky that may have been all you needed. But most of you will notice a few more things necessary. Yes, we have told sprite number 1 to go to position 50,50. But we haven't told it what shape to use. So let's do that now:

```
COMAL 0.14:  IDENTIFY 1,3
COMAL 2.0 :  IDENTIFY(1,3)
             SHOWSPRITE(1)
```

We just gave our sprite number 1 the shape stored as definition number 3. Notice that in 0.14, COMAL automatically shows the sprite. The COMAL Cartridge requires that you issue the SHOWSPRITE command (this is an advantage in many situations). While we are at it let's specify what color our sprite should be:

```
COMAL 0.14:  SPRITECOLOR 1,2
COMAL 2.0 :  SPRITECOLOR(1,2)
```

So we now have a red sprite. If your LOADSHAPE worked properly it should look like a red bat. Now, let's see the other two shapes we have stored in definitions 1 and 2. This will be quite easy. Remember, while in splitscreen mode, the top few lines are just a window onto the text screen. As we have been typing, this window has been moving along with us. Now, let's use the CURSOR UP key to move our text window up the text screen. Try it. Press CURSOR UP a couple times. You should see a few of the lines you entered just a minute ago. Use CURSOR UP and CURSOR DOWN to put the cursor on the IDENTIFY line. Now, simply use CURSOR RIGHT to glide over to the 3. It tells the sprite to use shape definition number 3. To see definition 2, simply type a 2 on top of the 3 and press return (the line now looks like this):

More ►

```
COMAL 0.14:  IDENTIFY 1,2
COMAL 2.0 :  IDENTIFY(1,2)
```

As soon as you pressed return, the shape used by our sprite number 1 was changed, and it instantly changed on the screen too. Great. Now look at image number 1 (use CURSOR UP and CURSOR RIGHT to change the 2 into a 1):

```
COMAL 0.14:  IDENTIFY 1,1
COMAL 2.0 :  IDENTIFY(1,1)
```

You've seen all three shapes now. But, let's make them bigger! Let's change the sprites size (of course COMAL has a word to do this):

```
COMAL 0.14:  SPRITESIZE 1,TRUE,TRUE
COMAL 2.0 :  SPRITESIZE(1,TRUE,TRUE)
```

Our sprite is immediately taller AND wider. To see a short but wide bat just change the last TRUE to FALSE (FALSE means don't expand the height).

Now, you may already be amazed at this lovely short wide red bat staring at you on your screen. But wouldn't it be much nicer if it would flap it's wings? Sure it would. So let's do it. And remember, we are still in direct mode:

```
COMAL 0.14:

FOR X=1 TO 300 DO IDENTIFY 1,X MOD 3+1

COMAL 2.0:

FOR X=1 TO 300 DO IDENTIFY(1,X MOD 3+1)
```

Your bat was really flapping wasn't it? Look at the line we used. It is a simple FOR loop, from 1 thru 300. Each time through the loop we issue an identify command. But rather than specify an exact image to use, we calculate it. Now you may begin to see how simple it is to have men running around the screen - or bats flying overhead. The MOD operator is perfect for our needs here. What MOD does is do a division, but throw away the answer. It then returns the REMAINDER as it's number. So here is how the calculations for our shape definition number went:

```
1 divided by 3 is 0 remainder 1.
Throw away the 0, use the remainder.
We now have 1.
Add 1 to it and we use definition 2.

2 divided by 3 is 0 remainder 2.
Throw away the 0, use the remainder.
We now have 2.
Add 1 to it and we use definition 3.

3 divided by 3 is 1 remainder 0.
Throw away the 1, use the remainder.
We now have 0.
Add 1 to it and we use definition 1.

4 divided by 3 is 1 remainder 1.
Throw away the 1, use the remainder.
We now have 1.
Add 1 to it and we use definition 2.

So the cycle goes ... 2..3..1..2..3...
```

And if we had been real computer people, we would have used definition numbers 0, 1 and 2 instead of 1, 2 and 3. Then we wouldn't have had to add 1 to each calculation. Sometimes it pays to be a computer person! But, I still like counting from 1, so we will stick with definitions numbered 1, 2, and 3.

Let's review what we just did. Press the F1 function key. That puts you back on the text screen. Don't worry. We haven't lost the graphics screen. Just press the F3 function key to see it again. F1 to go back to text screen.

Now, look at the text screen. There are all the things you typed in. Fantastic. Let's make them into a program, then next time we need only to type in RUN instead of all those lines. And you don't even have to retype them to convert them into program lines. To be part of a program, they only need a line number at the beginning.

More ►

The Guide

A Monthly Publication For Commodore™ Owners

Commodore support with a twist ... Personable and even humorous ... Timely news ... Helpful tutorials ... On-going support for several languages: BASIC (including BASIC 7.0 as featured in the 128 PC), Machine Language, COMAL, and Pascal ... Program Listings ... *Honest* software reviews.

The Guide features some of the best computer humorists to be found.

- Introduce your "widow" to the Computer Widow's Compendium.
- Tutorials and feature articles by the famous Mindy Skelton.
- Featuring Shelly Roberts' "I'm Sorry ... But I Don't Speak Hexidecimal." Discover why Shelly just may be the Andy Rooney of the computer world!

We feel we have assembled one of the most talented staff of writers in the Commodore world. Receive each month the most friendly and helpful Commodore publication available. Written by Commodore users who are writing to you, not down at you.

Limited offer — FREE With Each Subscription !

Subscribe *NOW* to capitalize on free software offer!

Receive up to $99.85 in *high quality* software!

With each year's subscription (or renewal) ordered, receive your choice of the software packages listed below, including the award winning educational games from Disney, or Omiterm Terminal written by our own Bob Richardson.

Subscription	Rate	Canadian *	Receive FREE
One Year	$18.00	$24.00 (U.S.)	Any software title listed below
Two Years	$35.00	$47.00 (U.S.)	Any two of the titles below
Three Years	$48.00	$64.00 (U.S.)	All three of the titles below

* Note: Canadian (and other foreign) subscribers must add $2 (U.S.) per program title to cover shipping and handling. No handling charge for U.S. subscribers. Canadian rates listed in U.S. dollars — send U.S. funds only, please.

Donald Duck's Playground

CES Software Showcase Award Winner! — Disney animation at its best! Children play four games to "earn" money to buy playground equipment. Builds money handling skills. Superb graphics. A bestseller! $39.95 retail value.

Winnie The Pooh In The Hundred Acre Wood

— Players explore the Hundred Acre Wood to find lost articles like Ow'ls books, Pooh's honey pot and Eeyore's tail, and return them to their rightful owners. Cheery music from the Disney movie caps off this computer rendition of the beloved classic. $39.95 retail value.

Omiterm Terminal Program

— Written by The Guide's own Bob Richardson. Fully supports the new 1660 Modem 300! (The *ONLY* commercial terminal package that currently fully supports the 1660!!!

- **Modem controls accessible from the keyboard**
- **Punter protocol — upload & download — 300/1200 baud**
- **Ten programmable function keys**
- **15 number phone directory**
- **20k receive buffer**
- **Tone or pulse dialing**
- **Auto dial/re-dial**
- **Half/full duplex**

A $19.95 retail value.

The Guide
3808 S.E. Licyntra Ct.
Portland, OR 97222

Don't delay — mail today! Supplies are limited. Offer expires August 31, 1985.

Name_____

Address_____

City, State & Zip_____

Enter My Subscription for:

☐ 1 Year ☐ 2 Years ☐ 3 Years

Please Check One: ☐ New ☐ Renewal

You are entitled to one FREE software package, as described above, for each year's subscription purchased. Please list your second choice, as supplies are limited on the Disney titles! Please allow six to eight weeks for delivery.

My first choice is:_____

My second choice is:_____

COMAL 0.14:

Remember, you have the LOADSHAPE procedure in the computer now (if not - you can get it back from the disk using the command: ENTER "LOADSHAPE.PROC"). But since its line numbers start at 9000 we have plenty of room in front of it.

COMAL 2.0:

You are lucky. LOADSHAPE is built into the cartridge. Don't worry about it.

Both versions:

You should have many of the lines we need in our program on your screen. Just add a line number in front of them. To add a line number use the cursor keys to put the cursor at the beginning of the line you need. Then just press SHIFT INST 5 times to insert 5 spaces. Then just type in the line number needed. Here is what your program should look like (your line numbers may vary - but the order of the lines should be the same):

COMAL 0.14:

```
100 LOADSHAPE(1,"SHAP.BAT1")
110 LOADSHAPE(2,"SHAP.BAT2")
120 LOADSHAPE(3,"SHAP.BAT3")
130 SETGRAPHIC 0
140 SPRITEPOS 1,50,50
150 IDENTIFY 1,1
160 SPRITECOLOR 1,2
170 SPRITESIZE 1,TRUE,FALSE
180 FOR X=1 TO 300 DO IDENTIFY 1,X MOD 3
    +1 // wrap line
```

COMAL 2.0:

```
95 USE SPRITES
100 LOADSHAPE(1,"SHAP.BAT1")
110 LOADSHAPE(2,"SHAP.BAT2")
120 LOADSHAPE(3,"SHAP.BAT3")
130 USE GRAPHICS
135 SPLITSCREEN
140 SPRITEPOS(1,50,50)
150 IDENTIFY(1,1)
155 SHOWSPRITE(1)
160 SPRITECOLOR(1,2)
170 SPRITESIZE(1,TRUE,FALSE)
180 FOR X=1 TO 300 DO IDENTIFY(1,X MOD 3
    +1)// wrap line
```

Notice the programs are essentially the same. With 2.0 you simply add parentheses around the parameters for all sprite commands. Plus before you can use sprites or graphics, you issue the USE command. Also, SPLITSCREEN is automatic in COMAL 0.14 but must be specified in 2.0. And remember the SHOWSPRITE line too.

OK. Now you have your program ready to try. SAVE it first - run it second:

```
SAVE "SPRITE'TEST"
RUN
```

We did it. A running program - or should I say a flying program. Yes, I know that the first IDENTIFY line is unnecessary since we use it in our FOR loop. But, it was already on the screen so I used it to make it simple. We just used all the lines on the screen.

Now, what we need is a disk full of sprite shapes for everyone to use. Remember, the shapes are the hard part. We are including some shapes on TODAY DISK #8. If you come up with some nice sprite SHAPE files, please send them in so we can share them with the others. Maybe we can put out a disk of just SPRITE SHAPES.

NOW FOR SOME ACTION ...

You have just seen how easy it is to set up and use a sprite. Now, since our readers are requesting COMAL games, I will show you how to move the sprites around the screen and even watch for collisions!

First let's move our sprite across the screen. The easy way is to use the program we already have. (I always say: steal from programs you already have.

More ►

Don't re-invent the wheel). That program should still be residing in your computer. Look at the FOR loop line:

COMAL 0.14:

```
FOR X=1 TO 300 DO IDENTIFY 1,X MOD 3+1
```

COMAL 2.0:

```
FOR X=1 TO 300 DO IDENTIFY(1,X MOD 3+1)
```

Instead of an IDENTIFY command after the DO, let's execute a procedure we will call MOVING and pass the current value of X as the parameter. You can retype the entire line, or do it the easy way:

* LIST the line on your screen.
* Move the cursor to the start of the word IDENTIFY.
* Type MOVING(X) on top of the IDENTIFY part. Type spaces to completely erase the rest of the old line.

Your new line in COMAL 0.14 and 2.0 looks like this:

```
FOR X=1 TO 300 DO MOVING(X)
```

Now we just write a procedure called MOVING. I recommend RENUMbering your program first, and then using large line numbers:

```
RENUM
AUTO 9000

PROC MOVING(NUM) CLOSED
 IDENTIFY 1,NUM MOD 3+1
 SPRITEPOS 1,NUM,50
ENDPROC MOVING
```

The above is for COMAL 0.14. In COMAL 2.0 just add the parentheses needed by IDENTIFY and SPRITEPOS.

Now, save your program:

```
SAVE "0:SPRITE-SAMPLE2"
```

RUN your program, and watch the bat fly across your screen.

Next, let's have TWO bats fly across the screen. To do this we only need to add 2 lines to our MOVING procedure. Add these lines just before your ENDPROC line.

If ENDPROC is line 9030 we use these:

```
9022 IDENTIFY 2,NUM MOD 3+1
9023 SPRITECOLOR 2,5 // green
9024 SPRITEPOS 2,NUM,150
```

COMAL 2.0 remember the parentheses plus:

```
9025 SHOWSPRITE(2)
```

RUN it and watch two bats fly across the screen. Notice how two different sprites (sprite 1 and sprite 2) both can use the same shape definitions.

For fun, let's have sprite number 1 gradually fly up towards sprite number 2. Then if they collide, have sprite number 1 turn white and fall down. First we add the lines to have sprite number 1 gradually move up:

CURRENT LINE:

```
SPRITEPOS 1,NUM,50
```

CHANGE IT TO:

```
SPRITEPOS 1,NUM,50+NUM
```

COMAL 2.0 remember to keep the parentheses. SAVE it:

```
SAVE "SPRITE-SAMPLE3"
```

RUN it and watch the lower sprite fly up, collide with the top sprite, and keep going right off the top of the screen. Now we will add the collision detection lines. Add a couple lines to your MOVING procedure so that it looks like this (new parts are in bold):

More ►

```
PROC MOVING(NUM)
 IDENTIFY 1,NUM MOD 3+1
 SPRITEPOS 1,NUM,50+NUM-FALLING
 IDENTIFY 2,NUM MOD 3+1
 SPRITECOLOR 2,5 // green
 SPRITEPOS 2,NUM,150
 IF SPRITECOLLISION(1,TRUE) THEN
  SPRITECOLOR 1,1 // white
  COLLIDED=TRUE
 ENDIF
 IF COLLIDED THEN FALLING:+2
ENDPROC MOVING
```

Look at these lines. (COMAL 2.0 remember the parentheses for the SPRITECOLOR statement.) We are using the variable COLLIDED to tell us if the sprites have collided. The variable FALLING is used to store the distance our sprite has fallen after colliding with the other sprite. But to properly use these variables, we have to initialize them at the start of the program. We will set FALLING to be 0 (no distance yet) and COLLIDED to be FALSE (no collision yet). Add this line to the start of your program:

```
FALLING=0; COLLIDED=FALSE
```

Look at the line that begins with SPRITECOLLISION. This is COMAL's built in collision detector. We simply tell it what sprite number to check and it will return TRUE if that sprite is touching another sprite, or FALSE if not. You noticed we used the number 1 for sprite number 1.

You also should notice the word TRUE as a second parameter. To understand what this means, you must first understand how the Commodore 64 keeps track of collisions (If you don't care about the behind the scenes details, just skip the next two paragraphs).

If a sprite collides, the computer changes the collision detector to TRUE for that sprite. It checks all 8 sprites. If a program asks if there has been a collision, the computer does two things, responds TRUE or FALSE, and resets the collision detector back to FALSE for all the sprites. If you want to check for collisions with more than one sprite, the computer ruined the chance to do so by changing the detector back to FALSE after the first one.

COMAL has corrected this problem for you. It can preserve the detector until you have checked for as many sprites as you wish. Then after you check all your sprites, it can reset the detector back to FALSE as it should be. But to do this, it needs to know when to reset the detector. That is what the second parameter is for. Use TRUE if it should reset the detector to FALSE or use FALSE if you wish to use the detector again. After you have checked all your sprites make sure that the detector is reset back to FALSE (just specify TRUE for your last sprite check).

Our program is now working. It demonstrates two moving sprites complete with collision detection. The program was presented in bits and pieces as we progressed. Here is the complete listing so you can verify that you typed everything in correctly. This program is also included on TODAY DISK #8.

```
// COMAL 2.0 Version
//  by Len Lindsay
//
USE graphics
USE sprites
falling:=0; collided:=FALSE
loadshape(1,"shap.bat1")
loadshape(2,"shap.bat2")
loadshape(3,"shap.bat3")
graphicscreen(0)
spritepos(1,50,50)
identify(1,1)
showsprite(1)
spritecolor(1,2)
spritesize(1,TRUE,FALSE)
FOR x:=1 TO 300 DO moving(x)
//
```

More ►

```
PROC MOVING(NUM)
 IDENTIFY(1,NUM MOD 3+1)
 SPRITEPOS(1,NUM,50+NUM-FALLING)
 IDENTIFY(2,NUM MOD 3+1)
 SPRITECOLOR(2,5) // green
 SPRITEPOS(2,NUM,150)
 SHOWSPRITE(2)
 IF SPRITECOLLISION(1,TRUE) THEN
  SPRITECOLOR(1,1)// white
  COLLIDED=TRUE
 ENDIF
 IF COLLIDED THEN FALLING:+2
ENDPROC MOVING
-------------------------------------
// COMAL 0.14 Version
falling:=0; collided:=false
loadshape(1,"shap.bat1")
loadshape(2,"shap.bat2")
loadshape(3,"shap.bat3")
setgraphic 0
spritepos 1,50,50
identify 1,1
spritecolor 1,2
spritesize 1,true,false
for x:=1 to 300 do moving(x)
//
proc loadshape(spr,filename$) closed
 dim ds$ of 2
 open file 81,filename$,read
 ds$:=status$
 if ds$="00" and spr>=0 and spr<64 then
  a:=828
  poke a+0,162 //    ldx #81
  poke a+1,81
  poke a+2,32 //     jsr $ffc6
  poke a+3,198
  poke a+4,255
  poke a+5,160 //    ldy #00
  poke a+6,0
  poke a+7,32 // lp  jsr $ffcf
  poke a+8,207
  poke a+9,255
  poke a+10,153 //   sta spr,y
  poke a+11,(49152+spr*64) mod 256
  poke a+12,(49152+spr*64) div 256
  poke a+13,200 //   iny
  poke a+14,192 //   cpy #64
  poke a+15,64
  poke a+16,208 //   bne lp
  poke a+17,245
  poke a+18,96 //    rts
  sys a
 endif
 close file 81
endproc loadshape
```

```
//
PROC MOVING(NUM)
 IDENTIFY 1,NUM MOD 3+1
 SPRITEPOS 1,NUM,50+NUM-FALLING
 IDENTIFY 2,NUM MOD 3+1
 SPRITECOLOR 2,5 // green
 SPRITEPOS 2,NUM,150
 IF SPRITECOLLISION(1,TRUE) THEN
  SPRITECOLOR 1,1 // white
  COLLIDED=TRUE
 ENDIF
 IF COLLIDED THEN FALLING:+2
ENDPROC MOVING ■
```

Shape Files 0.14

SAVESHAPE is a built in command in the COMAL 2.0 Cartridge Sprites Package. Now, COMAL 0.14 users can do the same thing. SAVESHAPE will save the shape definition specified (SPR) to a disk file named by FILENAME$. Simply type in the following procedure (use AUTO 9000 to get high line numbers) and LIST it to disk: LIST "SAVESHAPE.PROC". Then you can merge it into any future program. Then to save shape number 3 as a disk file named SHAP.BOAT3 issue this command:

```
SAVESHAPE(3,"SHAP.BOAT3")

proc saveshape(spr,filename$) closed
 dim ds$ of 2
 if spr>=0 and spr<64 then
  open file 81,filename$,write
  ds$:=status$
  if ds$="00" then
   for i#:=0 to 63 do
    byte:=peek(49152+spr*64+i#)
    print file 81: chr$(byte),
   endfor i#
  endif
  close file 81
 endif
endproc saveshape■
```

Remember What's His Name

or
Sorting by Sound

by Kevin Quiggle

Problem: You have a computerized mailing list with 1000 names on it and you're trying to find that guy whose name was Smith, or Smythe, or Smithson, or something like that. What do you do? You could try looking up every name that sounds familiar (and hope you get lucky), you could list all 1000 names and scan the list by hand (but then why have a computer!), or you could use a program that can search for similar sounding names. The last solution, a program that looks up names by "sound," is an appealing solution, so let's see if we can make it work.

Have we only traded one problem for another? How do we get our computer to recognize similar sounding names? An obvious solution would be to build a rhyming dictionary, and have the computer search for rhymes to the name we're looking for, but this would be a very large task indeed. Just putting together a rhyming dictionary would be a formidable job, and then we would have to figure out how to make it recognize rhymes for names like Smith, Trudeau, and Wojihowicz. A program capable of handling the job might require hundreds of thousands of bytes of memory.

There is a simpler solution, if we are willing to scale down our requirements a bit. Let's look for names that have similar spellings. With this idea in mind, we can make things even simpler by throwing out letters that have little effect on pronunciation, and combining the remaining letters into a few related groupings.

As you may have guessed by now, a method already exists to do just what we want. The U.S. Census Bureau had a problem much like the one described above, except that their "mailing list" included millions of names! The solution, now known as the Soundex algorithm, was actually developed by Margaret Odell and Robert Russell in 1918! Of course this was long before the advent of computers, but fortunately the method is easy to program, especially in a language like COMAL with excellent string handling capabilities.

The Soundex method works by producing a four character code (one letter, three numbers) for any name, in such a way that names that are spelled alike or sound alike produce identical codes. In outline form, the steps are as follows:

1) The first letter of the name is the first character of the Soundex code.

2) Where there are two or more identical adjacent letters, omit all but one.

3) From the rest of the name, eliminate all letters A,E,H,I,O,U,W,Y.

4) Encode the remaining letters as follows:

```
            B,F,P,V = 1
  C,G,J,K,Q,S,X,Z = 2
              D,T = 3
                L = 4
              M,N = 5
                R = 6
```

5) Form a four character code by dropping extra digits or adding trailing zeros. The following example shows how the names Odell and Russell would be encoded at each of the five steps:

More ►

```
1) Odell          Russell
2) Odel           Rusel
3) Odl            Rsl
4) O34            R24
5) O340           R240
```

Although the Soundex coding method sometimes produces identical codes for very different names, and different codes for similar names, it works remarkably well considering its elegant simplicity. The program listing below shows how to write the Soundex algorithm into COMAL not in thousands of bytes, but in a couple of hundred bytes! The program asks for a last name, converts the entry into all uppercase letters with procedure UPRC, and then produces and prints the Soundex code for the name entered. The Soundex procedure can easily be incorporated into any program, enabling you to produce and search for Soundex codes, instead of for exact matches. Two versions of the Soundex program are printed here for use with each version of COMAL. The 2.0 version uses some of the advanced features such as nested procedures and user defined string functions.

```
//  COMAL 0.14 version
//
print chr$(147)
dim code$ of 4, name$ of 20
input "Name? :": name$
uprc(name$)
soundex(name$,code$)
print name$;"=";code$
//
proc soundex(name$,ref code$) closed
 dim num$ of 18
 num$:="111122222222334556"
 code$:=name$(1)
 for i#:=2 to len(name$) do
  if name$(i#)<>name$(i#-1) then
   if not (name$(i#) in "AEHIOUWY") then
    code$:=code$+num$(name$(i#) in "BFPV
    CGJKQSXZDTLMNR") // wrap line
   endif
  endif
 endfor i#
 code$:=code$+"000"
endproc soundex
//
proc uprc(ref a$) closed
 for i:=1 to len(a$) do
  n:=ord(a$(i))
  if n>=65 and n<=90 then a$(i):=chr$(n
  +128) // wrap line
 endfor i
endproc uprc
```

```
// delete "soundex"
// save   "soundex"
//  by Kevin Quiggle
// COMAL 2.0 version
//
PAGE
DIM name$ OF 30
INPUT "Name? :": name$
PRINT soundex$(name$)
//
FUNC soundex$(name$) CLOSED
 DIM num$ OF 18, code$ OF 4
 uprc(name$)
 num$:="111122222222334556"
 code$:=name$(1)
 FOR i#:=2 TO LEN(name$) DO
  IF name$(i#)<>name$(i#-1) THEN
   IF NOT (name$(i#) IN "AEHIOUWY") THEN
    code$:+num$(name$(i#) IN "BFPVCGJKQ
    SXZDTLMNR") // wrap line
   ENDIF
  ENDIF
 ENDFOR i#
 code$:=code$+"000"
 RETURN code$
 //
 PROC uprc(REF a$) CLOSED
  FOR i:=1 TO LEN(a$) DO
   n:=ORD(a$(i))
   IF n>=65 AND n<=90 THEN a$(i):=CHR$(
   n+128) // wrap line
  ENDFOR i
 ENDPROC uprc
ENDFUNC soundex$ ■
```

Use of Names in COMAL

by Michael Erskine

I've been working in COMAL for about a year now. Happily, I never had a lot of experience with BASIC and therefore I am not having trouble with "BASIC thinking".

I don't presume to be a very good or experienced programmer, but I have seen enough programs to express certain feelings about correct habits when programming in COMAL.

The idea behind COMAL is to be able to write programs which describe the solution to the specific problem being solved and reflect the logical procedures (steps) involved in that solution. In the words of Mr. Christensen, "It is a fact not to be overlooked that programming languages are not only used to control computing machinery, but also for COMMUNICATION OF IDEAS." This is a very powerful and wonderful concept.

COMAL allows us to use up to 78 characters in a variable, procedure or function name. If we are to communicate ideas we must use words. The more descriptive and specific our names the better the distant reader of our programs will understand them. This is critical to his or her ability to use the program. A COMAL program should be so descriptive when it is read that no further documentation is necessary! Program flow is documented by forced indentation (upon listing); calculations and most tests should be isolated and identified by the use of functions; and procedures should be used whenever a section of code is used more than once.

The names used to describe these procedures, functions and variables should be very descriptive. In a procedure which names all the colors by assigning a numeric value to a name for each color, one should NOT assign variable names like BG=3 when he can say BLUEGREEN=3. As a consequence of the above naming we would have two possible statements to change the PENCOLOR at some later time in the program: PENCOLOR(BG) and PENCOLOR(BLUEGREEN). Which would you rather have to remember while you were writing the program? Which would you rather read if I had written the program?

In the same line of logic, why should I call a procedure to figure the standard deviation of a set of test scores something like SDEV(T()) when I could call it with a statement like:

```
STANDARD'DEVIATION(TEST'SCORES())
```

The naming facilities available in COMAL are designed by the authors of the language to support the already excellent names of their statements and commands.

The effective COMAL programmer will carefully select the names in order to describe the PROCedure, FUNCtion or variable AND it's use in the program.

He will also remember COMAL is NOT BASIC, not even enhanced BASIC. COMAL is COMAL! It's just better than anything else. Why try to describe a Porsche in terms of a Model-T?

Think structure! ■

2.0 Data Base Manager

by Robert Shingledecker

After creating a Data Base Program in COMAL 0.14 (see COMAL TODAY #6) I decided to try it again with the more powerful COMAL 2.0. Quite frankly I was impressed with all the power that the Cartridge afforded me in this endeavor. My thanks go to all my friends whose input and sugguestions for improvement help make this program what it is. A special thanks to my nephew Steve Smullen without whom this program may never have been written. For it was after many hours of arguing that we decided on what this program should be. Together we coded, tested, and debugged this system of programs. Also to Larry Phillips whose Screen Wizard program brought to my attention the getscreen and setscreen commands. The system is self prompting with help screens. Here is a quick overview:

Disk Preparation: Using standard procedures, FORMAT a disk for use as a data base disk. Next copy all 14 **db** files onto this disk. FORMAT a disk like this:

PASS "N0:DATA BASE,88" <RETURN>

Starting the Program: Place the data base disk just created into the disk drive type: **RUN "db'boot"** and press return. Use the HELP option.

Create DB (f2): You can have up to twenty fields per record. Records may contain up to 254 characters. The field name plus the field length must be able to be displayed on one line, i.e. maximum 40 characters total.

First enter the name of this data base. Next you will be given a blank screen to design your own input form. Use only the cursor keys to position the input prompts and use the special characters [ ] to indicate the field lengths. Only the first twenty lines may be used. When you are satisfied with your input form, press f7.

After your screen format is verified you need to tell it approximately how many records you expect to have. Be realistic! This is NOT an IBM mainframe! Keep the size small, usually less than 100 records works best. The system will handle up to 400 records but will take a long time to sort! The limit of 400 records is based on the number of free blocks with a maximum record size.

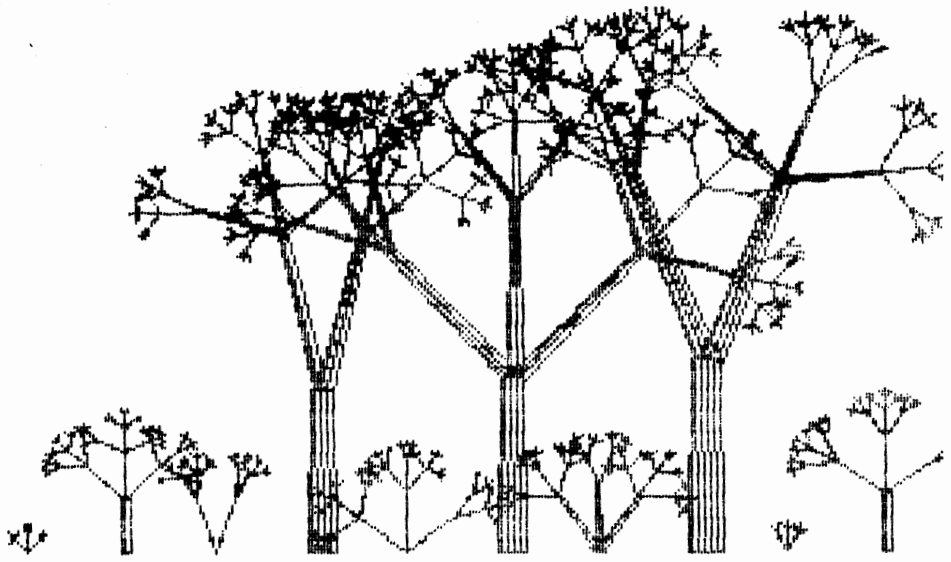
Now be patient as the system will set up the disk files for the dictionary and the data portions of your data base. You are allowed to add, delete, edit and display records in your database through the MAINTAIN DATABASE menu.

You also can search for a name or other group of characters. Enter the characters that should reside in the records you desire. The less you specify the more records that are likely to contain them and thus be selected. The more characters you specify the less records will be selected. Each record that contains the specified string, regardless in what field, will be selected for processing by the previously specified mode, (i.e., changing, deleting, displaying, or printing). However, if you have specified the use of an INDEX then the search is extremely fast and will search only the field that the index is defined for.

You also can print formatted reports as well mailing labels. Both can be in sorted order. The labels can include up to 9 fields on 5 printed lines.

The program is on TODAY DISK 8. ■

Random Forest



by Phred Linn

This is just a silly random forest program I wrote, based on the "tree" demo on the first COMAL disk I got. The programs for both 0.14 and 2.0 versions of COMAL are provided here.

COMAL 2.0 Version

```
// delete "0:trees"
// save   "0:trees"
//  by Phred Linn
//
USE graphics
graphicscreen(0)
background(1)
border(-1)
pencolor(0)
hideturtle
FOR y:=1 TO 5 DO
  moveto(y*60,0)
  length:=RND(10,70)
  ang:=RND(10,70)
  forest
  tree(length/3,RND(10,70))
  moveto(y*60-30,0)
ENDFOR y
//
PROC forest
  setheading(0)
  limb(2*INT(length/20)+4)
  tree(length,ang)
  moveto(y*60-30,0)
  tree(length/30,RND(10,70))
ENDPROC forest
//
PROC tree(length,ang)
  IF length<2 THEN RETURN
  left(ang)
  limb(2*INT(length/20))
  tree(length/2,RND(10,70))
  back(length)
  IF ang>25 THEN
    right(ang)
    limb(2*INT(length/20))
    tree(length/2,RND(10,70))
    back(length)
    right(ang)
    limb(2*INT(length/20))
    tree(length/2,RND(10,70))
    back(length)
```

More ►

```
  ELSE
    right(ang*2)
    limb(2*INT(length/20))
    tree(length/2,RND(10,70))
    back(length)
  ENDIF
  left(ang)
ENDPROC tree
//
PROC limb(times)
  forward(length)
  right(90)
  FOR x:=1 TO times STEP 2 DO
    forward(x)
    right(89)
    forward(length)
    right(91)
    forward(x+.035*length)
    right(91)
    forward(length)
    right(89)
  ENDFOR x
  forward(times/5)
  left(90)
ENDPROC limb
```

```
COMAL 0.14 Version

// delete "0:trees"
//  by phred linn
// save   "0:trees"
//
background 1
border 1
pencolor 0
setgraphic 0
hideturtle
for y:=1 to 5 do
 moveto y*60,0
 length:=rnd(10,70)
 ang:=rnd(10,70)
 forest
 tree(length/3,rnd(10,70))
 moveto y*60-30,0
endfor y
settext
end
```

```
//
proc forest
 setheading 0
 limb(2*int(length/20)+4)
 tree(length,ang)
 moveto y*60-30,0
 tree(length/30,rnd(10,70))
endproc forest
//
proc tree(length,ang)
 if length<2 then return
 left ang
 limb(2*int(length/20))
 tree(length/2,rnd(10,70))
 back length
 if ang>25 then
  right ang
  limb(2*int(length/20))
  tree(length/2,rnd(10,70))
  back length
  right ang
  limb(2*int(length/20))
  tree(length/2,rnd(10,70))
  back length
 else
  right ang*2
  limb(2*int(length/20))
  tree(length/2,rnd(10,70))
  back length
 endif
 left ang
endproc tree
//
proc limb(times)
 forward length
 right 90
 for x:=1 to times step 2 do
  forward x
  right 89
  forward length
  right 91
  forward x+.035*length
  right 91
  forward length
  right 89
 endfor x
 forward times/5
 left 90
endproc limb ■
```

C64 COMAL-80

by UniComal, ApS

COMAL is a semi-compiler. When lines are entered from the keyboard, they are checked for syntax errors. During this parsing the lines are translated into an internal code (Reverse Polish Notation).

For example, the expression:

```
1 + 3 * sin( - X / 2 ) + 3
```

is translated to reverse polish:

```
1 3 X 2 / - sin * + 3 +
```

Keywords and variable names are stored in "tokenized" format. That means that when you use the variable "FILE'NAME" the name is stored once (in the name table), and two bytes are used to refer to it (pointers into the name table).

COMAL works extensively with tables. A table for the KEYWORD names, KEYWORD routines (addresses), variable names, and variable information. The first two tables are built into COMAL, and the last two are built from the program.

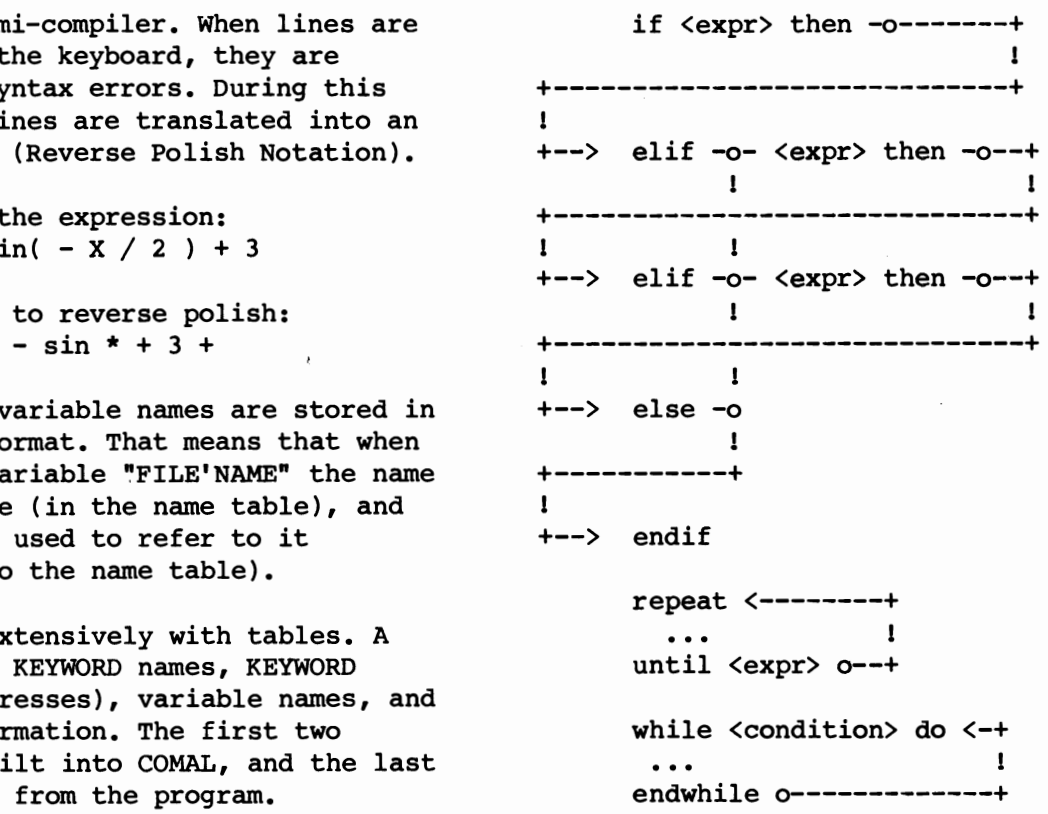
When you type in a line, or ENTER it from disk, the editor tokenizes the keywords, and checks each non-keyword against the name table. If the name is found, that token number is used. If the name is not found, a new entry is generated, and a token number assigned.

This allows fast program execution but also makes it harder to recreate (LIST) the original program lines.

When RUN is typed, a prepass is performed and the program is RUN. The prepass chains the program structures such as:

IF-ELIF-ELSE-ENDIF,
REPEAT-UNTIL
WHILE-ENDWHILE,

in this way:



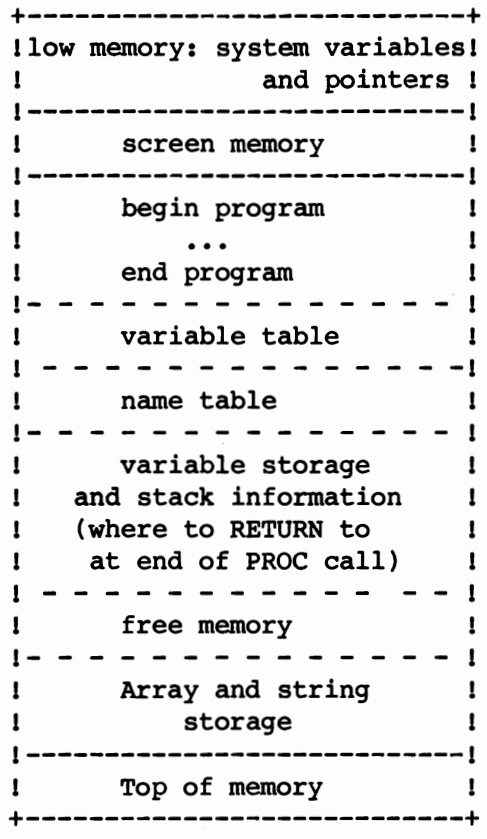
For example, after a THEN statement the absolute address of the next ELIF, ELSE or ENDIF is stored so that the system only has to figure out the <condition>, and not the branching. A special variant of this is the CASE statment where an address of zero is stored after the last WHEN if there is no OTHERWISE so that error detection is possible.

All comments are preceded with the address of the next valid statement so that no time is lost interpreting them. Also, PROC headers are preceded with the address of the statement past ENDPROC so that execution will jump around them if they are in the middle of program execution.

More ►

All structures are checked so that a REPEAT does not end with ENDIF and so on.

Programs are stored in memory in segments. Here is a generalized memory map for COMAL 2.0 (the 0.14 version is the same, with the language being between the screen memory and the beginning of the program):



When a call to a procedure is made, the return address is placed on the stack, as well as the addresses of the parameters (if the parameter of the PROC is not a REFerenced variable, a new variable is created, and the address of the new variable is used).

In COMAL 2.0, when an EXTERNAL procedure is called, the new procedure is placed at the start of free memory, so that no IMPORTing is possible. After the EXTERNAL procedure is loaded, it is SCANed and then run.

Because of the internal format of the program there are two types of program files:

A SAVE-file (prg-file) contains a program in the internal format. SAVE files can be LOADed and CHAINed. This is useful because it is fast to load and is compact.

A LIST-file (seq-file) contains the source text of a program. LIST-files can be ENTERed. This is useful when the program has to be transferred to another version of COMAL. It can also be used to handle procedure libraries.

This information is general in nature, and applies to both versions of COMAL for the Commodore 64. ■

Music Writer — 0.14

by David Stidolph

AN OVERVIEW

Although the Commodore 64 has wonderful sound capabilities, they have been under used due to the complexity involved. To help overcome this situation, the programs music'compiler, music'routine.mem (the source code is listed in music'routine.src), and music.proc are included on TODAY DISK 8 along with a sample song dance'hours.sng (a demo program called music/demo will play the song).

The idea behind the programs on the TODAY DISK is to allow you to develop complex, three voice music and to play it at the same time as a COMAL 0.14 program is executing.

This is accomplished by playing the music through the built in IRQ vector. Sixty times per second an interrupt is generated which is utilized by this music system.

WRITING MUSIC:

Using the built in COMAL screen editor, write the song as if it were a normal COMAL program, with comments at the start of each line, using the following format.

FORMAT:

The compiler is line oriented. Notes are entered like this:

```
0100 //[a4:8
      !!!!!!
      !!!!!+-- Duration / Eighth Note
      !!!!+--- Separator between octave
      !!!!       and duration
      !!!+---- Octave (0-7)
      !!+----- Note (a,b,c,d,e,f,g)
      !+------ Command ('[' means note)
      +------- First seven characters
                 are ignored
```

Before the compiler starts to decipher the line, it throws away the first seven characters (the line number, space, and double slash). In this case, this is what is left:

```
[a4:8
```

Where the first letter ([) shows the command (in this case a note line) and the rest defines the note, octave and duration. The line number and double slash marks at the beginning are to allow you to edit the program with the normal COMAL editor, and LIST it to disk. The music compiler ignores them and works on the rest of the line. A # after the note means it is sharp. Flats are not allowed.

From now on, I will assume you realize ALL commands are preceded with double slash (//) which allows COMAL to accept the line without check it for its syntax.

To play simultaneous voices separate the notes with a slash (/):

```
  b7:4/c#4:2/f2:4
   v1 /  v2 / v3
```

Note: to skip a voice put nothing where the note should be:

```
[//f2:4 assigns f2:4 to v3
[a0:1//e3:4 skips voice 2
```

More ►

COMMAND LIST

```
s       Set Voice Components
m       Master Control
%       Create Label
b       Branch to Label
c       Call Subroutine at Label
r       Return from Subroutine
t       Set Tempo
z       Zero Sid Chip
q       Turn Off Sid & Quit
[       Play Note(s)
;       Comment line
```

Command 'S' (SET)

```
 sw=t  set waveform to triangle
 sw=s  set waveform to sawtooth
 sw=p  set waveform to pulse
 sw=n  set waveform to noise
```

The following are set if x=1 and reset if x=0

```
 st=x  set/reset test bit
 sy=x  set/reset sync bit
 si=x  set/reset ring mode
 sf=x  set/reset filter on/off
```

The following are set to the value of x (x is from 0-15)

```
 sa=x  set attack rate
 sd=x  set decay rate
 ss=x  set sustain volume
 sr=x  set release rate
```

Pulse frequency is set to the value of x (x is from 0-4095)

```
 sp=x
```

To indicate which voice you want to change indicate it like a note with a slash '/'.

```
 sw=t/w=p/w=n
   v1/ v2/ v3
```

Sets waveform of voice 1 to triangle, voice 2 to pulse and voice 3 to noise.

To skip a voice put nothing between the slashes in its place.

```
s//r=12
```

Set the release value of voice 3 to 12.

Command 'M' (MASTER CONTROL)

Command 'm' is for master settings.

```
 mv=#    set volume (0-15)
 mf=#    set filter freq (0-4095)
 mr=#    set resonance (0-15)
 ml=#    set low pass filter (0-1)
 mh=#    set high pass filter (0-1)
 mb=#    set bandpass filter (0-1)
 m3=#    set voice 3 output (0-1)
 mx=#    set external voice filter (0-1)
```

Command '%' declare label

to declare label 'dixie':

```
 %:dixie
```

Labels are limited to eight characters. Nothing else is allowed on the line. Labels should be used ONLY where needed. The compiler has a limit on how many can be used.

Command 'B' BRANCH TO LABEL

To jump to a label 'dixie'

```
b:dixie
```

Nothing else is allowed on this line.

Command 'C' CALL SUBROUTINE

This command stores its current location and jumps to the location of the subroutine, regardless of where your song is in memory. To go to subroutine 'dixie'

```
 c:dixie
```

To return from a subroutine call, use the 'r' command.

Nothing else is allowed on this line.

More ►

Command 'R' RETURN FROM 'C'

This command does a RETURN from subroutine, just like the BASIC return statement does. As in BASIC, be careful you always call a music subroutine with the C command, and never have a subroutine as in-line music (jump around it if necessary), because if 'r' is executed without being called with the 'c' command the music player will crash.

Example: r

Command 'T' SET TEMPO

t=x where x is the length of a whole note in 1/60th seconds.

t=90
sets a whole note to 1.5 seconds.

Command 'Z' ZERO SID CHIP

z zero all registers

Command 'Q' QUIT

q zero all registers and quits playing music. (does not stop compiling)

Comment line ';'

; comment line

Anything after a semi-colin is ignored, so you can put comments or titles in front of sections of your music.

NOTES

All notes are first preceded on the line with the open bracket '[' character.

Rests can also be indicated. A rest will turn off the gate and pause that voice for the duration indicated.

[a#4:8/r:4/g3:2

This example has voice one play A sharp in the fourth octave for a duration of an eighth note, the second voice rests for a quarter note, and the third plays a G for a half note.

Dotted notes are notes which are to be 1 & 1/2 times its normal length. This is indicated with putting a single quote after the note duration.

[a#4:8' Means the note is the equivalent length of a eigth note and a sixteenth note.

A double dotted note is a note which is half again as long as a dotted note. This is indicated with a double quote (") after the duration.

[a#4:4" Means the note is the equivalent length of a quarter note plus an eigth note and a sixteenth note.

SUMMARY

This music system was designed to work with COMAL 0.14, but, due to problems with interfacing with COMAL, certain operations CANNOT be done while playing music. You cannot do any file access (reading files, CHAINING between programs, etc) or drawing on the graphics screen. If you do, the music will seem to drag (COMAL is blocking the interrupts).

A sample song is on TODAY DISK 8 titled "dance'hours.sng". This is a listed COMAL program done in the format explained above, ie - all comment lines. You can ENTER the file to see how a song is written.

To compile the song, LOAD and RUN the music compiler. Make sure you have expanded memory -- the compiler needs it all. The expand memory procedure is in the HI program on the 0.14 side of TODAY DISK 8. Make sure the disk in the drive has the file "music'player.mem" on it - the compiler LOADs this file first. Now just answer the questions.

More ►

The compiler will list each line as it works on it, and if there is an error, the program will tell what kind of error it is, and where it is located. When finished compiling, the song is played, and you are asked for a output file name to put the music data in. The compiler will then print the length of the music, and WRITE the music data string. To use this information, you use the procedures in "music.proc".

Any comments would be apreciated. Send them to me at **COMAL Users Group, USA.** ■

Function Keys

Larry L. Winckles of Holland, Ohio, Sent us a complete list of the Control Key functions for the COMAL 2.0 cartridge. An interesting feature of the Control U key is that it toggles the meaning of the f1, f3, and f5 function keys. Normally, these function keys give you their definitions (EDIT, USE TURTLE, and RENUM), but if you have used the command USE GRAPHICS, or USE TURTLE, these function keys are changed to show the textscreen (f1), splitscreen (f3), or the full graphics screen (f5). Control U will switch the meaning of the function keys and lock them into the new mode. The following procedure forces a Control 'U' into the keyboard buffer. This could be used right before the end of a program to cancel the graphics definition of the keys so the f1, f3, and f5 keys could be used for their normal meanings.

```
PROC toggle'graph'keys CLOSED
  POKE 631,21 // Control 'U'
  POKE 198,1
ENDPROC toggle'graph'keys
```
■

File Bug Fix

by Michael Erskine

How long will this one go around? There is a bug in the 1541 disk drive when writing relative files. COMAL is too fast for the drive. Previous notes about this problem were on page 17 of COMAL TODAY #7 and page 38 of COMAL TODAY #6.

While writing The Student Management Tool Kit last year, I ran into this little critter. Due to the nature of the records I was using, there was no way to accuratly predict exactly when the bug would be triggered without counting a lot of data. I was working with very large files consisting of lots and lots of short records, so counting data and jumping to a pause routine slowed the program more than was acceptable. I called EVERYONE: Colin, Len, Commodore, people from the User Group here in San Angelo, etc. It was hopeless.

Then I called David Stidolph and we spoke at considerable length about ways around the problem. One suggestion among several was to do a read after each write. It worked wonderfully and my problem was solved -- simply, easily and permanently! It seems this would be rather slow, and it is if you do not read from the block just written. If you read from the block just written, the read/write head in the drive does not need to move! It actually ran faster just doing a read than it did when it had a pause inserted!

Nearly all of my programs handle files and most use relative files. When I can, I close a file after using it (this is a good programming rule). If closing the file slows down the program too much, I read the record just written back into a dummy variable. It works for me! ■

Sound Routines for 0.14

by David Stidolph

The following procedures emulate some of the 2.0 sound commands. Memory locations 1018-1023 are used to store needed information (since the SID chip can only be written to - NOT read from).

The following parameter names are used in the procedures:

```
v   = Voice Number (1-3)
f   = Frequency (0-65535)
p   = Pulse Frequency (0-4095)
a   = Attack (0-15)
d   = Decay (0-15)
s   = Sustain (0-15)
r   = Release (0-15)
t   = Waveform ( 1 = Triangle
                 2 = Pulse
                 3 = Sawtooth
                 4 = Noise)
l   = Lowpass Filter (True / False)
b   = Bandpass Filter (True / False)
h   = Highpass Filter (True / False)
e   = External Voice
v1  = Voice One (On / Off)
v2  = Voice Two (On / Off)
v3  = Voice Three (On / Off)
fil= Filter Frequency (0 - 2047)
vol= Volume (0 - 15)

proc setfrequency(v,f)
 poke 54272+(v-1)*7,f mod 256
 poke 54273+(v-1)*7,f div 256
endproc setfrequency
//
proc pulse(v,p)
 poke 54274+(v-1)*7,p mod 256
 poke 54275+(v-1)*7,p div 256
endproc pulse
//
proc adsr(v,a,d,s,r)
 poke 54277+(v-1)*7,a*16+d
 poke 54278+(v-1)*7,s*16+r
endproc adsr
//
proc gate(v,on'off) closed
 temp:=int(peek(1020+v)/2)*2
 if on'off then temp:+1
 poke 1020+v,temp
 poke 54276+(v-1)*7,temp
endproc gate
//
proc sync(v,on'off) closed
 temp:=peek(1020+v)
 temp:=(temp div 4)*4+(temp mod 2)
 temp:+on'off*2
 poke 1020+v,temp
 poke 54276+(v-1)*7,temp
endproc sync
//
proc ringmod(v,on'off) closed
 temp:=peek(1020+v)
 temp:=(temp div 8)*8+(temp mod 4)
 temp:+on'off*4
 poke 1020+v,temp
 poke 54276+(v-1)*7,temp
endproc ringmod
//
proc soundtype(v,t) closed
 temp:=peek(1020+v)
 temp:=temp mod 16
 if t>0 then temp:+(2t-1))*16
 poke 1020+v,temp
 poke 54276+(v-1)*7,temp
endproc soundtype
//
proc filtertype(l,b,h,v3) closed
 temp:=peek(1019) mod 16
 temp:+v3*128+h*64+b*32+l*16
 poke 1019,temp
 poke 54296,temp
endproc filtertype
//
proc filterfreq(fil)
 poke 54293,fil mod 8
 poke 54294,fil div 8
endproc filterfreq
//
```

More ►

```
proc filter(v1,v2,v3,e) closed
 temp:=peek(1018)
 temp:=(temp div 16)*16
 temp:+v1+v2*2+v3*4+e*8
 poke 54295,temp
 poke 1018,temp
endproc filter
//
proc resonance(res) closed
 temp:=peek(1018)
 temp:=temp mod 16
 temp:+res*16
 poke 54295,temp
 poke 1018,temp
endproc resonance
//
proc volume(vol) closed
 temp:=peek(1019)
 temp:=(temp div 16)*16+vol
 poke 54296,temp
 poke 1019,temp
endproc volume ■
```

Sounds from Holland

The following two COMAL 2.0 procedures come from the Dutch COMAL Users Group. Although they are written in COMAL 2.0, you should be able to use them with the procedures above.

```
PROC siren CLOSED
  USE sound
  volume(15)
  setfrequency(3,80)
  soundtype(3,1)
  filtertype(0,0,0,1)
  soundtype(1,3)
  pulse(1,256)
  adsr(1,0,0,15,7)
  gate(1,1)
  frq:=8000
  FOR t:=1 TO 120 DO
    fq:=frq+osc3*3
    setfrequency(1,fq)
  ENDFOR t
  gate(1,0)
  FOR v:=15 TO 0 STEP -1 DO
    volume(v)
    FOR t:=1 TO 4 DO NULL
  ENDFOR v
ENDPROC siren
```

More ►

```
PROC alarm(num'beeps) CLOSED
  USE sound
  volume(15)
  adsr(1,7,0,8,0)
  soundtype(1,1)
  setfrequency(1,12837)
  FOR i:=1 TO num'beeps DO
    gate(1,1)
    FOR w:=1 TO 100 DO NULL
    gate(1,0)
    FOR w:=1 TO 50 DO NULL
  ENDFOR i
  volume(0)
ENDPROC alarm ■
```

Converting Decimal

One of the nice features of COMAL 2.0 is the ability to accept hexidecimal or binary numbers as well as normal decimal numbers. Decimal is BASE 10 notation. Hexidecimal is base 16, Binary is base 2, etc. Sometimes we need to output numbers not in decimal, but in some other notation. The following function will return a string containing the number you give it converted into the base you want. Notice that the function is very short. This is possible because COMAL allows recursion.

```
FUNC convert'base$(num,base) CLOSED
  DIM digit$ OF 36
  digit$:="0123456789abcdefghijklmnopqrs
  tuvwxyz" // wrap line
  IF num<base THEN
    RETURN digit$(num+1)
  ELSE
    RETURN convert'base$(num DIV base,ba
    se)+digit$((num MOD base)+1) //wrap
  ENDIF
ENDFUNC convert'base$
```

An example of how to use this function to print the Hexidecimal equivalent of 25 would be:

```
SCAN
PRINT "$",convert'base(25,16)
```

Which would print out:
$19

Fun Print for 2.0

by Ken Strahl

While learning to master the new string - handling commands of COMAL, I remembered some cute BASIC subroutines I had seen in a magazine that utilized BASIC's string - handling commands. After translating a few of these routines directly from BASIC to COMAL 2.0 procedures, I discovered that it was easier to completely rewrite the others in COMAL. In doing so, I got a little carried away and ended up adding a few procedures of my own that will spice up printed messages in your programs. COMAL string handling is so easy and straight-forward that I feel no other explanation is necessary other than to read the listing.

```
// delete "fun'print"
// save   "fun'print"
//  by Ken Strahl
//
// Some procedures adapted from
//  Commodore Magazine
//  Vol.4, No6, issue 27
//  article: "PRINTS CHARMING"
//  by Andy Gamble
//
// Reprinted with permission
//
USE system
USE graphics
textcolors(12,0,15)
PAGE
//
pyramid("THIS IS ONE EXAMPLE OF PYRAMID
PRINT") //wrap line
PRINT
//
wait(800)
textcolor(7)
hyper("This is an example of HYPER PRINT
") //wrap line
CURSOR currow+2,curcol
wait(800)
//
textcolor(6)
slide("This is a SLIDING message")
PRINT
wait(800)
//
textcolor(2)
crash("This word CRASH is better than BA
SIC's") //wrap line
wait(800)
//
PAGE
textcolor(4)
twinkle(10,5,"Press any key to stop the
TWINKLE") //wrap line
//
PAGE
textcolor(3)
bounce(10,4,"Press any key to stop the B
OUNCE") //wrap line
//
PAGE
textcolor(5)
parade(3,3,"This is an example of a word
 PARADE") //wrap line
//
textcolor(8)
slo'print(5,3,"And this is a SLO'PRINTed
 message") //wrap line
//
textcolor(10)
slow'under(7,3,"Watch the SLOW'UNDERlini
ng of this") //wrap line
//
textcolor(13)
fly(10,4,"The letters of this message FL
Y !!") //wrap line
//
textcolor(1)
alpha(12,2,"This is an example of ALPHA
print.") //wrap line
sparkle(14,2,"Finally press a key to end
 the SPARKLE") //wrap line
PRINT
END "                END OF DEMO"
//
```

More ►

```
PROC slide(message$) CLOSED
 USE system
 pad#:=(20-(LEN(message$) DIV 2))
 add$:=SPC$(pad#)+message$
 FOR char#:=LEN(add$)-1 TO 1 STEP -1 DO
  PRINT add$(char#:LEN(add$))
  CURSOR currow-1,curcol
  FOR delay#:=1 TO 50 DO NULL
 ENDFOR char#
 PRINT
ENDPROC slide
//
PROC crash(message$) CLOSED
 USE system
 FOR char#:=LEN(message$) TO 1 STEP -1
 DO //wrap line
  FOR place#:=1 TO (40-LEN(message$))
  DIV 2+char# DO //wrap line
   PRINT TAB(place#),message$(char#)
   CURSOR currow-1,curcol
  ENDFOR place#
 ENDFOR char#
ENDPROC crash
//
PROC hyper(message$) CLOSED
 USE system
 l#:=LEN(message$)
 IF l#/2<>l# DIV 2 THEN
  result$:=" "+message$
  l#:=LEN(result$)
 ELSE
  result$:=message$
 ENDIF
 FOR c#:=0 TO l#/2-1 DO
  PRINT ""145""
  PRINT AT currow,(20-c#): result$(l#/
  2-c#:l#/2), // wrap line
  PRINT result$(l#/2+1:l#/2+1+c#);
  PRINT ""145""
 ENDFOR c#
ENDPROC hyper
//
PROC slo'print(row#,col#,message$) CLOS
ED //wrap line
 FOR char#:=1 TO LEN(message$) DO
  FOR delay#:=1 TO 500 DO NULL
   PRINT AT row#,col#+char#: message$(c
   har#) //wrap line
 ENDFOR char#
 PRINT
ENDPROC slo'print
//
PROC pyramid(message$) CLOSED
 l#:=LEN(message$)
 IF l#/2<>l# DIV 2 THEN
  result$:=" "+message$
  l#:=LEN(result$)
 ELSE
  result$:=message$
 ENDIF
 FOR c#:=0 TO l#/2-1 DO
  PRINT TAB(20-c#),
  PRINT result$(l#/2-c#:l#/2),
  PRINT result$(l#/2+1:l#/2+1+c#)
  ENDFOR c#
ENDPROC pyramid
//
PROC slow'under(row#,col#,message$) CLO
SED //wrap line
 line$:+""183""
 FOR char#:=0 TO LEN(message$) DO
  line$:+""
 ENDFOR char#
 PRINT AT row#,col#: message$
 FOR char#:=1 TO LEN(message$) DO
  PRINT AT row#+1,col#: line$(1:char#)
  FOR delay#:=1 TO 300 DO NULL
 ENDFOR char#
ENDPROC slow'under
//
PROC alpha(row#,col#,message$) CLOSED
 USE system
 FOR place#:=1 TO LEN(message$) DO
  m#:=ORD(message$(place#))
  IF (m#<65 OR m#>90) OR (m#<193 OR m#>
  218) THEN //wrap line
   PRINT AT row#,col#+place#: CHR$(m#)
  ENDIF
  IF m#>192 AND m#<219 THEN
   FOR letter#:=193 TO m# DO
    PRINT AT row#,col#+place#: CHR$(let
    ter#),""29"" //wrap line
   ENDFOR letter#
  ELIF m#>64 AND m#<91 THEN
   FOR letter#:=65 TO m# DO
    PRINT AT row#,col#+place#: CHR$(let
    ter#),""29"" //wrap line
   ENDFOR letter#
  ENDIF
  PRINT CHR$(13)
 ENDFOR place#
ENDPROC alpha
//
```

More ►

```
PROC twinkle(row#,col#,message$) CLOSED
 length#:=LEN(message$)
 DIM dum#(length#)
 FOR t#:=1 TO length# DO
  dum#(t#):=FALSE
 ENDFOR t#
 PRINT AT row#,col#: message$
 WHILE KEY$<>CHR$(0) DO NULL
 WHILE KEY$=CHR$(0) DO
  ran#:=RND(1,length#)
  IF dum#(ran#)=FALSE THEN
   PRINT AT row#,(col#+ran#)-1: ""18"",
   PRINT message$(ran#)
   dum#(ran#):=TRUE
  ELSE
   PRINT AT row#,(col#+ran#)-1: message
   $(ran#) //wrap line
   dum#(ran#):=FALSE
  ENDIF
 ENDWHILE
ENDPROC twinkle
//
PROC bounce(row#,col#,message$) CLOSED
 length#:=LEN(message$)
 DIM dum#(1:length#)
 FOR t#:=1 TO length# DO
  dum#(t#):=FALSE
 ENDFOR t#
 WHILE KEY$<>CHR$(0) DO NULL
 PRINT AT row#,col#: message$
 WHILE KEY$=CHR$(0) DO
  ran#:=RND(1,length#)
  IF dum#(ran#)=FALSE THEN
   PRINT AT row#-1,col#+ran#-1: message
   $(ran#) //wrap line
   PRINT AT row#,col#+ran#-1: " "
   dum#(ran#):=TRUE
  ELSE
   PRINT AT row#-1,col#+ran#-1: " "
   PRINT AT row#,col#+ran#-1: message$(
   ran#) //wrap line
   dum#(ran#):=FALSE
  ENDIF
 ENDWHILE
ENDPROC bounce
//
PROC wait(jiffy#) CLOSED
 FOR delay:=1 TO jiffy# DO NULL
ENDPROC wait
//
PROC fly(row#,col#,message$) CLOSED
 length#:=LEN(message$)
 FOR letter#:=1 TO length# DO
  FOR temp#:=1 TO 8 DO
   flag1#:=RND(1,2)
   flag2#:=RND(col#,length#+col#)
   IF flag1#=1 THEN
    PRINT AT row#-1,flag2#: message$(le
    tter#) //wrap line
    FOR delay:=1 TO 25 DO NULL
     PRINT AT row#-1,flag2#: " "
   ELSE
    PRINT AT row#+1,flag2#: message$(le
    tter#) //wrap line
    FOR delay:=1 TO 25 DO NULL
     PRINT AT row#+1,flag2#: " "
   ENDIF
  ENDFOR temp#
  PRINT AT row#,col#: message$(1:letter
  #) //wrap line
 ENDFOR letter#
ENDPROC fly
//
PROC parade(row#,col#,message$) CLOSED
 length#:=LEN(message$)
 FOR letter#:=1 TO length# DO
  FOR loc#:=1 TO (col#-1)+letter# DO
   PRINT AT row#+1,loc#: message$(lette
   r#) //wrap line
   FOR delay:=1 TO 10 DO NULL
   PRINT AT row#+1,loc#: " "
  ENDFOR loc#
  PRINT AT row#,col#: message$(1:letter
  #) //wrap line
 ENDFOR letter#
ENDPROC parade
//
PROC sparkle(row,col,message$) CLOSED
 USE system
 USE graphics
 bordc:=inq(1); backc:=inq(2)
 textc:=inq(3)
 WHILE KEY$<>CHR$(0) DO NULL
 PRINT AT row,col: message$
 WHILE KEY$=CHR$(0) DO
  CURSOR currow-1,1
  textcolors(bordc,backc,RND(0,15))
  PRINT AT row,col: message$
 ENDWHILE
 textcolors(bordc,backc,textc)
ENDPROC sparkle ■
```

Transposition

by Larry Phillips

(Reprinted from the newsletter of the Commodore Computer Club of Vancouver BC, CANADA)

The scene opens in a company cafeteria. John is seated at one of the tables eating his lunch, and browsing through a Stereo equipment magazine. Bill and Dave enter, seating themselves at the same table.

BILL: Oh! Is that the new Farkle Mark IV digital tweeter system?

JOHN: Yes, I'm thinking of getting one to go with my Instrumental Bass Systems Model six horizontally polarized mid range augmenter.

BILL: Personally, I wouldn't have any IBS equipment. I really think the Common Door equipment is just as good, and costs a lot less.

JOHN: Sure, but there aren't any high band wobbulators available for it. It's just not a serious stereo system.

DAVE: You guys and your stereos! How much money have you spent on them?

JOHN: About 7 or 8 thousand, but it's really good stuff. Well worth it.

BILL: Well I only spent half that much, and mine is pretty good.

DAVE: Eight thousand? Wow! I couldn't justify a tenth of that. I mean what could it do for me? How could it make money for me? Could I use it to do practical things around the house?

JOHN: You don't understand, Bill, it's fun to play with. It's relaxing to listen to. Besides, you could make money with good equipment by setting up a recording studio, or by becoming a stereo consultant.

BILL: Sure, and what's more practical than providing yourself with pleasure and entertainment.

DAVE: Alright, assume that I could justify a purchase on those grounds, which system should I buy? There are so many to choose from, and the cheap ones seem to be just for playing Disco. I would probably want one that could play more serious music.

JOHN: Well the IBS system would be perfect for you. It will play all the serious stuff, and besides, they use CP/M, you know, Compatible Playing for Music. That means that there is an enormous record and tape base available for the IBS.

BILL: Just a minute John, the Common Door system may not have quite as much soundware available, but it has everything you need. On top of that, the Common Door has a lot of nice features built in that you don't have on your system. Things like full song editing and less restrictive song titles.

DAVE: Listen, all this sounds like APL to me. Which is the best system?

JOHN: Just don't go with the Common Door stuff, it's only good for playing Disco. Get yourself an IBS, you'll never regret it.

More ►

BILL: Don't listen to that garbage, Dave. Of course there is serious stuff available for the Common Door. The thing he hasn't told you is that there isn't ANY Disco available for the IBS.

The discussion grows into an argument, and the only thing that all agree on is that one should never, under any circumstances, buy the Audio Stack system. Dave leaves the cafeteria with the hint of a smile on his face. He enjoys getting those two started. He also secretly owns a stereo marketed by a well known watch manufacturer, and wouldn't dream of admitting it.

(Ed.note- you might want to read this again. Fine stories, like fine wine, age well.) ■

Reiteration

by David Tamkin

The program below is a simple recursion vs. reiteration race across the Euclidean algorithm, suitable for entering into either COMAL 0.14 or 2.0. It includes two functions, GCD (greatest common divisor) being calculated recursively and GCF (greatest common factor) calculated iteratively. The results are inconclusive: both methods give correct answers, but sometimes one is faster, sometimes the other, depending on the two numbers input, and on the order in which they are given. One Thursday on PlayNet there was a bit of a row on recursion vs. reiteration, so you decide which manner of coding you like better. COMAL gives us a choice.

```
// delete "0:recursion'test"
//  by David W. Tamkin, June 18, 1985
// save   "0:recursion'test"
//
//Euclidean algorithm benchmark
//
repeat
 print chr$(147),chr$(14),
 print "Enter two natural numbers",
 input ": ": one,another
 if one>=1 and another>=1 then
  poke 162,0
  print gcf(one,another);peek(162)
  poke 162,0
  print gcd(one,another);peek(162)
 endif
until one<1 or another<1
print "See for yourself."
end
//
func gcd(larger,smaller) closed
 if smaller>larger then
  swap:=smaller; smaller:=larger
  larger:=swap
 endif
 if smaller<1 then
  print "Argument error for GCD"
  stop
 endif
 if larger mod smaller then
  return gcd(smaller,larger mod smaller)
 else
  return smaller
 endif
endfunc gcd
//
func gcf(larger,smaller) closed
 if smaller>larger then
  swap:=smaller; smaller:=larger
  larger:=swap
 endif
 if smaller<1 then
  print "Argument error for GCF"
  stop
 endif
 while larger mod smaller do
  swap:=smaller
  smaller:=larger mod smaller
  larger:=swap
 endwhile
 return smaller
endfunc gcf
```
■

Recursion

by Dick Klingens

One of the COMAL features is the possibility to use recursion mainly for evaluation of functions.

Looking at:

```
fac(5)=5 * 4 * 3 * 2 * 1
```

we find:

```
fac(5)=5 * fac(4)
```

In general:

```
fac(n)=n * fac(n-1) if n>1
fac(1)=1
```

This concept when translated into a recursive function in COMAL would be:

```
FUNC fac(n) closed // or not
  IF n=1 then
    RETURN 1
  ELSE
    RETURN n*fac(n-1)
  ENDIF
ENDFUNC fac
```

For the Greatest Common Divisor we get:

```
FUNC gcd(a,b)
  IF a>b then
    RETURN gcd(a-b,b)
  ELIF a<b THEN
    RETURN gcd(a,b-a)
  ELSE
    RETURN a
  ENDIF
ENDFUNC gcd
```

In both cases recursion is not necessary:

```
FUNC fac(n) CLOSED // now necessary
  f:=1
  FOR t:=n TO 1 STEP -1 DO f:=f*t
  RETURN F
ENDFUNC fac
```

and

```
FUNC gcd(a,b)
  REPEAT
    IF a>b THEN
      a:=a-b
    ELIF b>a THEN
      b:=b-a
    ENDIF
  UNTIL a=b
  RETURN a
ENDFUNC gcd
```

Recursion can also be used in non mathematical applications. Looking at the common structure of a process: input, processing, output, we have the main program:

```
//
input'
processing
output'
//
```

and after that the procedure:

```
PROC input'
  read'data
  ok:=FALSE
  control'data
  IF NOT ok THEN input'
ENDPROC input'
```

and see, recursion again.

There are, however, functions just asking for recursion. One of those is the so called Ackermann function defined by

More ►

```
if p=0      a(p,q) = q + 1
if q=0             = a(p-1,1)
if p<>0 and q<>0   = a(p-1,a(p,q-1))
```

In the following table we find the evaluation of A(2,1). The table contains also a list of the used function arguments.

Step	Evaluation	Arguments
0	a(2,1)	2 1
1	a(1,a(2,0))	1 2 0
2	a(1,a(1,1))	1 1 1
3	a(1,a(0,a(1,0)))	1 0 1 0
4	a(1,a(0,a(0,1)))	1 0 0 1
5	a(1,a(0,2))	1 0 2
6	a(1,3)	1 3
7	a(0,a(1,2))	0 1 2
8	a(0,a(0,a(1,1)))	0 0 1 1
9	a(0,a(0,a(0,a(1,0))))	0 0 0 1 0
10	a(0,a(0,a(0,a(0,1))))	0 0 0 0 1
11	a(0,a(0,a(0,2)))	0 0 0 2
12	a(0,a(0,3))	0 0 3
13	a(0,4)	0 4
14	5	5

In COMAL

```
FUNC a(p,q)
  IF p=0 THEN
    RETURN q+1
  ELIF q=0 THEN
    RETURN a(p-1,1)
  ELSE
    RETURN a(p-1,a(p,q-1))
  ENDIF
ENDFUNC a
```

BY placing the used arguments in an array we are able to compute the values without recursion. To do this we must build up and break down the list of arguments according to a fixed algorithm, until the list consist of only one argument: the wanted value.

We have:

```
DIM arg(150)
INPUT "X,Y = ": x,y
t:=2 // inital number of arguments
arg(1)=x; arg(2):=y; step':=0
out
REPEAT
  compute(arg(t-1),arg(t))
  out
UNTIL t=1
//
PROC compute(p,q)
  step':+1
  IF p=0 THEN
   t:-1; arg(t):=q+1
  ELIF q=0 THEN
    arg(t-1):=p-1; arg(t):=1
  ELSE
    t:+1
    IF t>150 then halt
    arg(t-2):=p-1; arg(t-1):=p
    arg(t):=q-1
  ENDIF
ENDPROC compute
//
PROC out
  PRINT step'
  FOR k:=1 TO t DO PRINT arg(k);
  PRINT
ENDPROC out
//
PROC halt
  PRINT "OVERFLOW"
  END
ENDPROC halt
```

With this program it is possible to compute values for the arguments X=0,1,2,3 and Y=0,1,2,3,4. Example: A(3,4) equals 125 in 1037 steps and A(4,0) equals 13 in 107 steps.

We can prove that the maximum number of elements used in the array equals the function value. That is why DIM ARG(150) is sufficient. A(4,1)=65532 and would take a long time as well as exceed the memory limit of our computer. ■

Useful Procedures

The following two procedures can be used ONLY by COMAL 2.0 (however, they could be modified for COMAL 0.14). One will print ANY sequential file to the screen while the shift key is depressed. Printing will continue only as long as the shift key is held down. The other procedure will change the disk device number. This is useful for people who have multiple disk drives.

```
PROC change'dev(old,new) CLOSED
  DIM f$ OF 20
  f$:="u"+STR$(old)+":m-w"
  f$:+""119""0""2""+CHR$(new+32)
  f$:+CHR$(new+64)+"/d-/s15"
  OPEN FILE 250,f$,WRITE
  CLOSE FILE 250
ENDPROC change'dev
```

```
PROC type(filename$) CLOSED
  USE system
  TRAP
    OPEN FILE 79,filename$,READ
    WHILE NOT EOF(79) DO
      PRINT GET$(79,254),
      WHILE PEEK(653)=0 DO NULL // wait
    ENDWHILE // for shift key to print
    CLOSE FILE 79
  HANDLER
    CLOSE FILE 79
    bell(3)
    PRINT ERRTEXT$
  ENDTRAP
ENDPROC type ■
```

Font Sprite Maker

by Len Lindsay

This program was whipped together in one evening at the Lincoln College Commodore Computer Camp. The program can make a set of 6 sprite shapes out of any character. The character can even be part of a user defined font. Try it first with a normal letter (use 1 for A). Then try other characters. COMAL 2.0 only.

```
PAGE
DIM c$ OF 8, a$ of 8, s$ of 64
DIM array$(1:64) OF 48
USE font
USE sprites
//
INPUT "character num (1-255/1=a):":char
INPUT "font to use (0-1 user / 2-3 reg)
: ": fontset //wrap line
PRINT "Now setting up the sprite shapes"
init'array
getcharacter(fontset,char,a$)
FOR x:=1 TO 8 DO
  c$:=a$(x:x)
  b$:=bin$(ORD(c$))
  FOR y:=1 TO 8 DO
    IF b$(y:y)="1" THEN setdot(x,y)
  ENDFOR y
ENDFOR x
setup'sprites
showit
END ""
//
FUNC bin$(n) CLOSED
  DIM res$ OF 8
  res$:="00000000"
  bit:=1
  FOR i#:=8 TO 1 STEP -1 DO
    IF n BITAND bit THEN res$(i#:i#):="
    1" //wrap line
    bit:+bit
  ENDFOR i#
  RETURN res$
ENDFUNC bin$
//
PROC setdot(row,col)
  start'row:=(row-1)*8+1
  start'col:=(col-1)*6+1
  FOR cur'row:=start'row TO start'row+7
   DO //wrap line
    FOR cur'col:=start'col TO start'col
    +5 DO //wrap line
      array$(cur'row)(cur'col):="1"
    ENDFOR cur'col
  ENDFOR cur'row
ENDPROC setdot
//
PROC init'array
  FOR rows:=1 TO 64 DO
    FOR cols:=1 TO 48 DO
      array$(rows)(cols):="0"
    ENDFOR cols
  ENDFOR rows
ENDPROC init'array
//
PROC setup'sprites
  FOR rows:=0 TO 2 DO
    FOR cols:=0 TO 1 DO
      sprite'number:=rows*2+cols+1
      s$:=""
      start'row:=rows*21+1
      start'col:=cols*24+1
      FOR dotrow:=start'row TO start'ro
      w+20 DO //wrap line
        FOR dotcol:=start'col TO start'
        col+23 STEP 8 DO //wrap line
          s$:+CHR$(VAL("%"+array$(dotro
          w)(dotcol:dotcol+7))) //wrap
        ENDFOR dotcol
      ENDFOR dotrow
      define(sprite'number,s$+"0")
    ENDFOR cols
  ENDFOR rows
ENDPROC setup'sprites
//
```

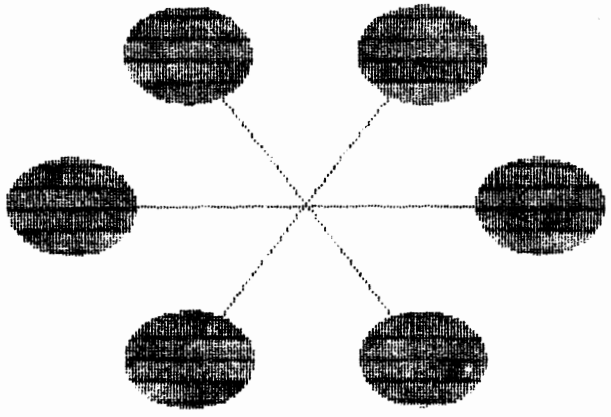
More ►

```
PROC showit
  USE graphics
  graphicscreen(0)
  show'row:=3*21*2; show'col:=1
  FOR dd:=1 TO 6 DO spritecolor(dd,char
   MOD 16) //wrap line
  FOR x:=1 TO 6 DO
    identify(x,x)
    showsprite(x)
    spritesize(x,TRUE,TRUE)
  ENDFOR x
  FOR moving:=show'col TO show'col+100
  STEP 2 DO //wrap line
    FOR sprite'row:=0 TO 2 DO
      FOR sprite'col:=0 TO 1 DO
        sprite'number:=sprite'row*2+spr
        ite'col+1 //wrap line
        spritepos(sprite'number,moving+
        sprite'col*48+1,show'row-sprite
        'row*42+1) //wrap line
      ENDFOR sprite'col
    ENDFOR sprite'row
  ENDFOR moving
ENDPROC showit ■
```

SHAPE DATA STATEMENT MAKER

While writing the sprite tutorial article for this issue, I needed a program that could be listed with the article that would create three different FLYING BAT images. We already had the shapes defined in disk files. We needed a quick way to convert those files into DATA statements. Of course, with COMAL it was no problem, and the program to do just that only took a few minutes to write. The program listing follows:

```
// save   "make'data"
//  by COMAL TODAY staff
// COMAL 2.0 version
PAGE
DIM file'from$ OF 20
DIM file'to$ OF 20
//
PRINT "This program will make DATA"
PRINT "statements from a shape file"
PRINT
INPUT "Shape name: ": file'from$
INPUT "Data Statment file: ": file'to$
PRINT
open'file
convert
PRINT "You can now MERGE the DATA"
PRINT "statements with any program."
END "" // no ending message
//
PROC open'file
  TRAP
    MOUNT
    OPEN FILE 1,file'from$,READ
    OPEN FILE 2,file'to$,WRITE
  HANDLER
    CLOSE
    PRINT ERRTEXT$
    END "" // no other end message
  ENDTRAP
ENDPROC open'file
//
PROC convert
  line:=1000; element:=0
  max'per'line:=8
  WHILE NOT EOF(1) DO
    num:=ORD(GET$(1,1))
    element:+1
    IF element=1 THEN
      PRINT FILE 2: line;"DATA";num,
    ELSE
      PRINT FILE 2: ",",num,
    ENDIF
    IF element=max'per'line THEN
      line:+10; element:=0
      PRINT FILE 2: // end line
    ENDIF
  ENDWHILE
  CLOSE
ENDPROC convert ■
```



Converting Sprite Master Files

by Captain COMAL

The following program will convert sprite image files created by Access Software's **SPRITE MASTER** into COMAL shape files. The program will ask what SPRITE MASTER block number you want (up to 10 per disk numbered 1-10), read in the images from that file, allow you to swap disks, and write each image into a seperate shape file for use with COMAL.

```
// delete "0:sprite'master"
// save   "0:sprite'master"
//  by Captain COMAL
//
dims
get'input'name
read'data
wait'second
get'output'name
write'data
END
//
PROC wait'second
  PRINT
  PRINT "Take out old disk, put in new
  disk" //wrap line
  wait'key
ENDPROC wait'second
//
PROC get'output'name
  PRINT
  PRINT "Enter file output name:";
  buffer("shap.")
  INPUT "": file[out]$
ENDPROC get'output'name
//
PROC dims
  DIM sprite$(16) OF 64
  DIM file[out]$ OF 18
  DIM file[in]$ OF 18, num$ OF 10
  DIM out$ OF 18
ENDPROC dims
//
PROC get'input'name
  PRINT CHR$(147),CHR$(14),
  PRINT "Sprite Master File Converter"
  PRINT
  PRINT "This program will convert
  PRINT "images created with Access
  PRINT "Sprite Master to COMAL shape f
  iles." //wrap line
  PRINT
  PRINT "Insert disk with sprite master
   files" //wrap line
  wait'key
  REPEAT
    PRINT "Enter sprite master block#(1
    -10):"; //wrap line
    INPUT "": block
    block:=INT(block)
    PRINT
  UNTIL block>0 AND block<11
ENDPROC get'input'name
//
PROC read'data
  file[in]$:="spritedata "
  num$:="0123456789"
  IF block=10 THEN
    file[in]$:=file[in]$+"10"
  ELSE
    file[in]$:=file[in]$+num$(block+1)
  ENDIF
  PASS "i0"
  OPEN FILE 2,file[in]$,READ
  sp'num:=0
  WHILE NOT EOF(2) DO
    sp'num:+1
    INPUT FILE 2: color
    IF color=0 THEN color:=128
    INPUT FILE 2: front
    INPUT FILE 2: mult0
    INPUT FILE 2: mult1
    FOR x:=1 TO 64 DO
      INPUT FILE 2: byte
      sprite$(sp'num)(x):=CHR$(byte)
    ENDFOR x
    sprite$(sp'num)(64):=CHR$(color-128)
  ENDWHILE
  CLOSE FILE 2
ENDPROC read'data
```

More ►

```
//
PROC write'data
  PASS "i0"
  FOR x:=1 TO sp'num DO
    IF x>9 THEN
      out$:=file[out]$+"1"+num$((x MOD
      10)+1) //wrap line
    ELSE
      out$:=file[out]$+num$(x+1)
    ENDIF
    OPEN FILE 2,out$,WRITE
    PRINT FILE 2: sprite$(x),
    CLOSE FILE 2
  ENDFOR x
ENDPROC write'data
//
PROC wait'key
  PRINT "and press any key to continue"
  WHILE KEY$<>CHR$(0) DO NULL
  WHILE KEY$=CHR$(0) DO NULL
  PRINT
ENDPROC wait'key
//
PROC buffer(a$) CLOSED
  l:=LEN(a$) MOD 11
  FOR x#:=1 TO l DO
    POKE 630+x#,ORD(a$(x#))
  ENDFOR x#
  POKE 198,l
ENDPROC buffer ■
```

Julian Dates

The following procedure and function will translate our standard way of dating (month, day, year) to the Julian calendar (which counts only the number of days since the first day of the year; ie 1-365). They were based on an idea by David Eagle. These are written for COMAL 2.0, but could be adapted to 0.14 if error trapping wasn't required.

```
func jdate(m,d,y) closed
  restore days'per'month
  total:=0
  trap
    for i#:=1 to m-1 do
      read days
      total:+days
    endfor i#
    read days
    if d<=days then
      total:+d
    else
      report
    endif
    odd:=y mod 4
    if odd and m>2 then
      total:-1
    elif odd and m=2 and d=29 then
      report
    endif
    return total
  handler
    return -1
  endtrap
  //
days'per'month:
  data 31,29,31,30,31,30
  data 31,31,30,31,30,31
endfunc jdate

proc cdate(jdate,ref m,ref d,y) closed
  restore days'per'month
  total:=0; m:=0
  trap
    while total<jdate do
      read days
      m:+1
      if m=2 and (y mod 4) then days:=28
      total:+days
    endwhile
    total:-days
    d:=jdate-total
  handler
    m:=-1; d:=-1
  endtrap
  //
days'per'month:
  data 31,29,31,30,31,30
  data 31,31,30,31,30,31
endproc cdate ■
```

More ►

Goof-Proof Programs

When writing a program, make sure you remember the person you are writing it for. If you write a program only for yourself, little documentation or error trapping is needed. If, on the other hand, it is for others to use, it is necessary to document the program so they know what to enter, and when to enter it. This is usually done through PRINT statements and INPUT prompts, but can also be in written instructions with specific examples. Look at the documentation from the user's point of view.

Error trapping is when a program checks for possible errors. This includes checking to see if a certain file is on the disk drive before it is used and checking INPUTed data for its accuracy before using it. Whenever important information is entered, it is a good idea to have a "last chance" option before the data is written to disk or printed out.

The following function can be used to check if a file exists:

```
func file'exists(name$) closed
 dim ds$ of 2
 open file 79,name$,read
 ds$=status$
 close file 79
 if ds$="00" then
  return true
 else
  return false
 endif
endfunc file'exists
```

Because COMAL 2.0 constantly checks for errors during disk access, the following procedure could be used instead:

```
FUNC file'exists(name$) CLOSED
  TRAP
    OPEN file 79,name$,read
    CLOSE FILE 79
    RETURN TRUE
  HANDLER
    CLOSE FILE 79
    RETURN FALSE
  ENDTRAP
ENDFUNC file'exists
```

ERROR HANDLER

The COMAL 2.0 Cartridge has extensive error handling abilities. Use its TRAP - HANDLER structure for error prone areas. The following is a list of commands to consider putting within the TRAP - HANDLER section of code.

CHAIN, COPY, CREATE, DIR, GET$, INPUT, MOUNT, OPEN, PASS, PRINT FILE, READ, READ, RENAME, SELECT INPUT, SELECT OUTPUT, VAL, WRITE

Using the Cartidge Error Handler you can write programs which do not crash when a user accidently types in a name instead of a number. Look at the following example:

```
LOOP
  TRAP
    INPUT "Enter Zip Code": zip
    EXIT
  HANDLER
    PRINT "Numbers Only!!!"
  ENDTRAP
ENDLOOP
// Rest of program
```

More ►

This simple loop will accept numbers only. Any non-numeric entry will cause the program to switch to the PRINT statment within the HANDLER - ENDTRAP section of code. If no error is encountered, the program continues to the EXIT statement and leave the loop. This loop is rather simplistic; it could also check to make sure all input is valid.

The TRAP - HANDLER structure lets a programmer anticipate errors, allow for them, and still keep the program readable.

Another feature of COMAL 2.0 is the protected INPUT field. Let's take our earlier example of INPUTing a number. Since USA Zip Codes are only 5 digits, you could use the following code to make sure that no more than 5 digits were entered:

```
LOOP
 TRAP
  INPUT AT 4,1,5:"Enter Zip Code ": zip
  EXIT
 HANDLER
  PRINT AT 2,1: "Numbers Only Please."
  PRINT AT 4,1: SPC$(39) // clear line
 ENDTRAP
ENDLOOP
// Rest of program
```

The statement INPUT AT is usually used to move the cursor to a certain spot and start the INPUT there. In this case we use **4,1** as the starting row & column coordinates (for row 4 column 1). It will the print **Enter Zip Code**, and wait for the user to type characters and press the RETURN key. The **,5** following the **4,1** tells COMAL that the INPUT field (or area) is only **5** characters long. The cursor up/down keys are ignored, as are keys that could affect the screen outside the INPUT area. Now a carefully designed screen will not be destroyed with the CLR/HOME key. The following is a general purpose function that will return an INPUT from the user:

```
FUNC get'input$(prompt$,default$,row,col
,length) CLOSED // wrap line
  z:=ZONE
  ZONE 0
  DIM string$ OF length
  l'd:=LEN(default$); l'p=LEN(prompt$)
  PRINT AT row,col:prompt$,"[",
  PRINT default$,SPC$(length-l'd),"]",
  PRINT AT row,col+l'p+1:"",
  INPUT AT 0,0,length:"": string$,
  ZONE z
  RETURN string$
ENDFUNC get'input$
```

You could use a function like this to ask questions of the user. The brackets immediately tell the user exactly how long the INPUT area is, and the default value can be used if the user does not quite understand what is going on. The following lines would use this function:

```
d$:="sample file"
p$:="Enter file name: "
filename$:=get'input$(p$,d$,5,1,16)
```

The user would see the following on row number 5 of the screen with the cursor blinking on the **s** of 'sample':

```
Enter file name: [sample file     ]
```

In this case, if the return key were simply pressed, the string variable **filename$** would be set to "sample file".

There is much more to cover in handling errors, but this is a start. Anticipate errors! Compensate for possible problems! When you're done hand it to someone who has never seen it before and see if they understand it. ■

Programming Batch Files — 2.0

by Ian MacPhedran

The batch files provided by the COMAL 2.0 cartridge are extremely useful programming aids. The following ramblings are meant to be a series of tips for the programmer, which can be used to assist in batch file use.

MAKING A BATCH FILE:

The method which I use to create a batch file requires a wordprocessor capable of producing a text (SEQ) file on disk (I use SpeedScript 3.0 from COMPUTE! magazine). It is handy if the wordprocessor allows you to embed control characters in the text file (For example cursor keys, clear screen, character colour keys, etc.), as these characters will be printed to the screen and produce the same effect as they normally would if typed on the keyboard.

LOOPING IN A BATCH FILE:

Batch files cannot branch from one line to another. Therefore, looping must be handled in a special way. While some progam control statements (e.g. REPEAT, GOTO, multiline WHILE and FOR, etc.) cannot be used, single line FOR and WHILE statements, as well as single line IF statements can be used. Also, multiline REPEAT ... UNTIL loops can be emulated in the following manner:

```
(BAT.FILE'ONE)
...
x:=20  // set counter x
SELECT INPUT "0:bat'loop" //call loop

(BAT'LOOP)
...  // loop contents
x:-1 // count down
IF x>0 SELECT INPUT "0:bat'loop"
// if not zero loop back to start
... // if zero continue in bat'loop
```

INTERACTING WITH BATCH FILES:

Because the batch file takes over in place of the keyboard, the user cannot communicate with the computer via this medium. However, interaction can be achieved by using joysticks, or the shift key flag (a batch file can't press a shift key). The following portion demonstrates the use of the shift keys:

```
...
// PRESS SHIFT TO CONTINUE [BLACK]
a:=0
WHILE a:=0 DO a:=PEEK($28d)
[WHITE]
...
```

Note that this assumes that the background colour is black. [BLACK] stands for an embedded character (control 1), and [WHITE] stands for control 2. The shift key flag location returns a 1 if the shift key is pressed, a 2 if the Commodore key is pressed, and a 4 if the control key is pressed. Thus, the variable "a" can be used to select another action after the loop, if desired.

BATCH FILE TEXT PRESENTATION:

All of the batch file's text is presented on the text screen. Therefore, information can be passed to the user in the form of comment lines (lines starting with a ! or //). These lines will be ignored by the interpreter, but will be displayed on the screen. The above example shows the comment line asking the user to press a shift key to continue. However, it may not always be desired to show all of the instructions to the user. Control characters are important in this situation. By setting the text color to the background color (which can also be

More ►

set), the text is hidden against the screen colour. If the programmer is not able to embed special characters in his/her text file, an alternate method would be to cover the text screen with the graphics screen. The following shows an example of this method.

```
USE system
textcolors(0,0,1) // White on Black
PAGE
(WHT)
! This is text which comes onto the
! screen. If a special character is
! used (BLK)the text is hidden.
! (WHT)Press any key to continue(BLK)
...
```

BATCH FILE DATA TRANSFER:

Batch files can also be used to pass data to a program. This can be helpful if the user is running an interactive program, or if data is required by a program, and may be entered by either keyboard or data file.

First we create the batch file on the disk in the disk drive. Type in the following lines in the order they appear (use either METHOD A or METHOD B):

Method A:

```
SELECT OUTPUT "bat.sample"
PRINT "123"
PRINT "678,80"
SELECT OUTPUT "ds:"
```

or Method B:

```
OPEN FILE 2,"bat.sample",WRITE
PRINT FILE 2: "123"
PRINT FILE 2: "678,80"
CLOSE FILE 2
```

Both of the two methods above will create the same batch file and can be used from direct mode or as part of a program. Of course, you should only use one of them.

These four lines will create the batch file that can be used in the following program. Type the program in using AUTO to create the line numbers for you.

```
DIM r$ of 1, f$ of 16
f$:="bat.sample"
REPEAT
  INPUT "Use file (y<cr>/n)": r$
UNTIL a$ IN "yYnN"
IF r$ IN "yY" THEN SELECT INPUT f$
INPUT a,b,c
SELECT INPUT "kb:"
<more code>
```

This COMAL program can accept the input from the batch file, and use it as if it were input from the keyboard.

MODIFYING PROGRAMS:

David Stidolph's article in COMAL Today 7 shows how a program may be modified using batch files. I can only add that, in some cases, EXTERNAL PROCs may be better than modifying the program (self-modifying programs are considered bad programming practice in most cases).

HOUSEKEEPING TASKS

Batch files can be used as housekeeping programs, cleaning up disks that are full, NEWing programs after use, restoring or changing system defaults, and other handy tasks. The coding for such a system is demonstrated in the following example. Again, we must first create the batch file before we can use it.

```
SELECT OUTPUT "bat.cleaner"
PRINT "NEW // erase old program"
PRINT "USE system"
PRINT "textcolors(14,6,14) //set color"
PRINT "PAGE // clear screen"
SELECT OUTPUT "ds:"
```

To use this batch file, add the following line before any END statement:

```
SELECT INPUT "bat.cleaner"
```

Depreciation

by John F. Eldredge

This 2.0 program calulates and prints depreciation schedules, using the three most common methods: Straight-line, declining-balance, and sum-of-years-digits. Output may be sent to either the screen or the printer.

Since the floating-point arithmetic used by COMAL gives nine significant decimal digits, the largest starting value which the program will accept is $9,999,999.99. If you use a starting value of $1,000,000 or larger, check the results for minor rounding errors (floating-point math is done in binary, not decimal).

The number-printing procedure, PRINT'DOLLARS, is an improvement over the PRINT USING command, since it supplies a "floating" dollar sign and optional commas.

```
// delete "depreciation"
// save   "depreciation"
// by John F. Eldredge
//
initialize
REPEAT
 get'values
 REPEAT
  print'choices
  REPEAT
   choice$:=KEY$
  UNTIL choice$ IN ""133""134""135""136"
" //wrap line
  IF choice$=""133"" THEN // f1
   straight'line
  ELIF choice$=""134"" THEN // f3
   declining'balance
  ELIF choice$=""135"" THEN // f5
   sum'of'yrs'digits
  ENDIF
 UNTIL choice$=""136"" // f7
UNTIL FALSE // loop forever
//
PROC initialize
 USE graphics
 DIM depreciation(20), choice$ OF 1
 DIM name$ OF 32
 PAGE
 PRINT ""14"",
 background(12)
 border(12)
ENDPROC initialize
//
PROC get'values
 PAGE
 PRINT TAB(10),
 PRINT ""5"Depreciation Schedules"
 PRINT
 PRINT ""144"Maximum starting value:";
 PRINT "$9,999,999.99"
 PRINT "Maximum life of asset: 20 years"
 PRINT "Do not use commas in numbers"
 PRINT
 REPEAT
  INPUT "Starting value: ": starting'va
  lue //wrap line
 UNTIL starting'value<100000000 AND sta
 rting'value>0 //wrap line
 REPEAT
  INPUT "Salvage value: ": salvage'value
 UNTIL salvage'value<starting'value AND
 salvage'value>=0 //wrap line
 REPEAT
  INPUT "Life of asset (2 to 20 years):
  ": life# //wrap line
 UNTIL life#<21 AND life#>1
 PRINT "Send output to screen or printe
 r (s/p)?" //wrap line
 REPEAT
  choice$:=KEY$
 UNTIL choice$ IN "sp"
 use'printer:=(choice$="p")
ENDPROC get'values
//
```

More ►

```
PROC print'choices
 PRINT ""147"",
 PRINT TAB(12),""5"Choose a method:"
 PRINT
 PRINT "f1"144" Straight-line depreciat
 ion" //wrap line
 PRINT ""5"f3"144" Declining-balance de
 preciation" //wrap line
 PRINT ""5"f5"144" Sum-of-years-digits
 depreciation" //wrap line
 PRINT
 PRINT TAB(19),""5"or"
 PRINT
 PRINT "f7"144" Enter new values"
ENDPROC print'choices
//
PROC straight'line
 book'value:=starting'value
 FOR year#:=1 TO life#-1 DO
  depreciation(year#):=round((startin
  g'value-salvage'value)/life#) //wrap
  book'value:=book'value-depreciation
  (year#) //wrap line
 ENDFOR year#
 depreciation(life#):=book'value-salvag
 e'value //wrap line
 name$:="STRAIGHT-LINE DEPRECIATION"
 print'schedule
ENDPROC straight'line
//
PROC declining'balance
 POKE 631,50 //ASCII for "2"
 POKE 198,1 //1 char. in buffer
 INPUT "Depreciate at normal rate times
 : ": rate //wrap line
 rate:=rate/life#
 book'value:=starting'value
 FOR year#:=1 TO life# DO
  depreciation(year#):=round(book'value
  *rate) //wrap line
  IF (book'value-depreciation(year#))<s
  alvage'value THEN //wrap line
    depreciation(year#):=book'value-sal
    vage'value //wrap line
  ENDIF
  book'value:=book'value-depreciation(y
  ear#) //wrap line
 ENDFOR year#
 name$:="DECLINING-BALANCE DEPRECIATION"
 print'schedule
ENDPROC declining'balance
//
PROC sum'of'yrs'digits
 sum#:=0
 FOR year#:=1 TO life# DO sum#:=sum#+y
 ear# //wrap line
 book'value:=starting'value
 FOR year#:=1 TO life#-1 DO
  depreciation(year#):=(starting'value-
  salvage'value)*(life#-year#+1)/sum#
  //wrap line
  depreciation(year#):=round(depreciati
  on(year#)) //wrap line
  book'value:=book'value-depreciation(y
  ear#) //wrap line
 ENDFOR year#
 depreciation(life#):=book'value-salvag
 e'value //wrap line
 name$:="SUM-OF-YEARS-DIGITS DEPRECIAT
 ION" //wrap line
 print'schedule
ENDPROC sum'of'yrs'digits
//
PROC print'schedule
 IF use'printer THEN
  //open file 255,"",unit 4,7,write
  SELECT OUTPUT "lp:"
  PRINT TAB((80-LEN(name$))/2+1),name$
  PRINT
  PRINT TAB(19),"STARTING VALUE:",
  PRINT TAB(38),
  print'dollars(starting'value,TRUE)
  PRINT
  PRINT
  PRINT TAB(41),"ACCUMULATED"
  PRINT "      YEAR        DEPRECIA",
  PRINT "TION         ",
  PRINT "DEPRECIATION          ",
  PRINT "BOOK VALUE"
  PRINT
  book'value:=starting'value
  accum'depr:=0
  FOR year#:=1 TO life# DO
   accum'depr:=accum'depr+depreciation(
   year#) //wrap line
   book'value:=book'value-depreciation(
   year#) //wrap line
   PRINT TAB(8),
   PRINT USING "##": year#,
   PRINT TAB(17),
   print'dollars(depreciation(year#),TR
   UE) //wrap line
   PRINT TAB(38),
   print'dollars(accum'depr,TRUE)
```

More ►

```
   PRINT TAB(59),
   print'dollars(book'value,TRUE)
   PRINT
  ENDFOR year#
  PRINT
  PRINT TAB(19),"SALVAGE VALUE     ",
  print'dollars(salvage'value,TRUE)
  PRINT
  SELECT OUTPUT "ds:"
  CLOSE FILE 255
 ELSE
  PRINT ""147""5"",
  PRINT TAB((40-LEN(name$))/2+1),name$
  PRINT "STARTING VALUE:"144"",
  print'dollars(starting'value,FALSE)
  PRINT
  PRINT ""5"YEAR  DEPREC'N  ACCUM. DEPR
  . BOOK VALUE"144"" //wrap line
  book'value:=starting'value
  accum'depr:=0
  FOR year#:=1 TO life# DO
   accum'depr:=accum'depr+depreciation(
   year#) //wrap line
   book'value:=book'value-depreciation(
   year#) //wrap line
   PRINT USING " ##": year#,
   print'dollars(depreciation(year#)
   ,FALSE) //wrap line
   print'dollars(accum'depr,FALSE)
   print'dollars(book'value,FALSE)
   PRINT
  ENDFOR year#
  PRINT ""5"SALVAGE VALUE: "144"",
  print'dollars(salvage'value,FALSE)
  PRINT
 ENDIF
 PRINT TAB(9),"PRESS "5"RETURN"144" TO
 CONTINUE", //wrap line
 WHILE KEY$<>CHR$(13) DO NULL
ENDPROC print'schedule
//
FUNC round(number)
 RETURN INT((number+5e-03)*100)/100
ENDFUNC round
//
PROC print'dollars(number,use'commas) C
LOSED //wrap line
 DIM dollar$ OF 14
 dollar$:="              "
 sign:=SGN(number)
 number:=INT((ABS(number)+5e-03)*100)
 FOR i#:=12 TO 4 STEP -1 DO
  dollar$(i#):=CHR$(number MOD 10+48)
  number:=number DIV 10
 ENDFOR i#
 dollar$(1:9):=dollar$(2:10)
 dollar$(10):="."
 i#:=3
 WHILE dollar$(i#) IN " 0" DO
  dollar$(i#):=" "
  i#:=i#+1
 ENDWHILE
 dollar$(i#-1):="$"
 IF sign<0 THEN dollar$(i#-2):="-"
 IF use'commas THEN
  dollar$:="  "+dollar$
  IF dollar$(8) IN "0123456789" THEN
   dollar$(1:7):=dollar$(2:8)
   dollar$(8):=","
   IF dollar$(4) IN "0123456789" THEN
    dollar$(1:3):=dollar$(2:4)
    dollar$(4):=","
   ENDIF
  ENDIF
 ENDIF
 PRINT dollar$,
ENDPROC print'dollars ■
```

DUTCH ERROR MESSAGES

On each side of the COMAL Today Disk 8 is a file containing Dutch Error Messages. The 2.0 file is called "pkg.dutch'errors", and you load the error messages with the following commands typed in direct mode:

```
LINK "pkg.dutch'errors"
USE dutch
```

To return to English you issue the command: USE english.

The 0.14 file is called "dutch'errors". To use it you copy this file onto a disk and then rename it "comalerrors". Make sure that the old "comalerrors" file is not on the disk, or is deleted before you rename the Dutch error file.

1541 Directory Editor

by Robert Ross

This program is a directory editor for all 1541 compatible drives (1541, MSD, Indus, etc). It will allow you to read a disk directory into memory, manipulate it (sort, add, delete entries, protect / unprotect entries, etc), and write the updated directory back onto the disk. This will not change the programs or data on the disk, only the directory. **Try this program only on test disks until you are familiar with its operation.**

NOTE: some letters stand for different options when shifted.

OPTIONS:

'r' read directory of disk in unit 8, drive 0
'L' list a range of entries
'l' list up to current entry
'+' add 1 to current entry
'-' subtract 1 from current entry
'C' set new curren entry
'i' make space to insert entries
'd' delete entries (old entries saved at top)
'W' write entries, 1 to high, to unit 8, drive 0
'e' edit an entry
's' swap 2 entries
'c' copy one entry to another
'S' sort entries
'p' protect range of entries
'u' unprotect range of entries
'z' zero a range of entries
'h' set high entry
'H' help screen
'RUN/STOP' usually stops program

24 data bytes are stored for each entry.

SORTING

Sorting may be done on from 1 to all 24 bytes of the entries. Sort direction (><) may be set for each byte: '>' means 'a' will preceed 'b' after the sort. 'width' (1-24) is the # of characters used for comparing entries. '1st' and 'last' give the entry sort range.

'specify' allows the order and direction of sorting to be set; the letters a to x stand for the 24 bytes:

[a=file type; b,c=track,sector of first block; d-s=file name; t,u=track,sector for the first side sector of a relative file; v=relative file record length; w,x=low byte,high byte of file length].

e.g.:'width'=1, order="a...", direction="<..." would sort by decreasing file type groups.

'names' sets a default order with d-s at the start and all directions = '>'

PROTECTING PROGRAMS

You can protect programs from being scratched or deleted with the 'p' option. To do only one entry, give the same number for the first and the last of the range identifier. You can unprotect them with the 'u' option.

The program is on TODAY DISK 8. ■

CORRECTIONS

The article entitled "Sharing" in COMAL Today #7 was written by Norm Pollinger. (It was sent to us with "author unknown") A belated thanks to Norm!

Speedscript SEQ Converter

by Kendall French & Phyrne Bacon

The advantage of converting files produced by Speedscript to sequential files is that almost all Commodore-64 word processors can load sequential files. For example, whenever I submit a program to COMAL TODAY, I write the program comments in Speedscript, save them on disk, and then use SPEED'TO'SEQ to convert them into a sequential file. [Ed note: many wordprocessors such as Easy Script and Paper Clip can write a SEQ file directly, avoiding the extra conversion necessary with Speedscript].

SPEED'TO'SEQ converts Speedscript files into sequential files, and SEQ'TO'SPEED converts sequential files into Speedscript files. With a dual disk drive, the drive to be read from or written to can be specified by using the prefix 0: or 1: with the filename.

The following COMAL 0.14 program will covert a Speedscript Text file into a SEQuential file. A version for COMAL 2.0 using some of its advanced features is also included.

```
// delete "0:speed'to'seq"
// save   "0:speed'to'seq"
//  version 2.0
//
PAGE
begin
poke'machine'language
convert'file
goodbye
//
PROC begin
  DIM name$ OF 20, inputname$ OF 22
  DIM outputname$ OF 16, ckey$ OF 1
  DIM ccursor$(0:1) OF 4
  CLOSE
  PRINT ""8""14""
ENDPROC begin
//
PROC convert'file
  center(""17"Convert Speedscript to Se
  q") //wrap line
  REPEAT
    error:=0
    get'name(""17"Enter Speed text file
     name") //wrap line
    inputname$:=name$+",p,r"
    openfile(2,inputname$)
    CLOSE FILE 2
  UNTIL NOT error
  // until file is open properly
  REPEAT
    error:=0
    get'name("Enter Seq file name")
    outputname$:=name$+",s,w"
    openfile(3,outputname$)
  UNTIL NOT error
  PRINT FILE 3: ""
  OPEN FILE 2,inputname$
  PRINT
  center("Converting the file...")
  SYS 828 //machine language
  center(""17"Closing the file")
  CLOSE
ENDPROC convert'file
//
PROC poke'machine'language
  total:=0
  FOR loc:=828 TO 947 DO
    READ y
    total:+y
    POKE loc,y
  ENDFOR loc
  IF total<>15815 THEN
    PRINT "ERROR in DATA statements!!"
    goodbye
  ENDIF
  DATA 162,2,32,198,255,160,0,32
  DATA 207,255,153,0,159,165,144,41
  DATA 64,208,6,200,208,241,169,0
  DATA 44,169,255,133,253,132,252,32
  DATA 204,255,160,0,185,0,159,32
  DATA 148,3,153,0,159,200,208,244
  DATA 162,3,32,201,255,160,0,185
```

More ►

```
  DATA 0,159,240,3,32,210,255,165
  DATA 253,208,10,200,208,241,32,204
  DATA 255,162,0,240,179,196,252,240
  DATA 4,200,208,227,0,76,204,255
  DATA 201,31,176,4,105,64,208,22
  DATA 201,31,208,4,169,13,208,14
  DATA 201,64,144,10,201,91,176,4
  DATA 105,128,208,2,169,0,96,0
ENDPROC poke'machine'language
//
PROC goodbye
  CLOSE
  END ""9""
ENDPROC goodbye
//
PROC get'char
  ccursor$(0):="> "157""157""
  ccursor$(1):="< "157""157""
  n:=0
  d:=0
  REPEAT
    PRINT ccursor$(d),
    d:=NOT d
    REPEAT
      ckey$:=KEY$
      n:=n+1
      IF n>60 THEN n:=0
    UNTIL n=0 OR ckey$<>CHR$(0)
  UNTIL ckey$<>CHR$(0)
ENDPROC get'char
//
PROC get'name(title$)
  center(title$)
  PRINT "      "17"[",
  name$:=""
  REPEAT
    get'char
    s:=ORD(ckey$)
    IF ckey$=CHR$(20) AND LEN(name$)>0
    THEN //wrap line
      name$:=name$(1:LEN(name$)-1)
      PRINT ckey$,
    ENDIF
    IF (s>31 AND s<96) OR (s>192 AND s<
    224) THEN //wrap line
      name$:=name$+ckey$
      PRINT ckey$,
    ENDIF
  UNTIL ckey$=CHR$(13)
  PRINT "]"17""
  IF name$="" THEN goodbye
ENDPROC get'name
PROC center(message$)
  PRINT SPC$(20-LEN(message$)/2),messag
  e$ //wrap line
ENDPROC center
//
PROC openfile(num,name$)
  TRAP
    MOUNT
    OPEN FILE num,name$
  HANDLER
    CLOSE FILE num
    PRINT ERRTEXT$;name$
    error:=TRUE
  ENDTRAP
ENDPROC openfile

-----------------------------------
// delete "0:speed'to'seq"
//  by Phyrne Bacon
// save   "0:speed'to'seq"
//  version 0.14
//
print chr$(147)
poke'machine'language
begin
convert'file
goodbye
//
proc begin
 dim name$ of 20, inputname$ of 20
 dim outputname$ of 20
 dim ccursor$(0:1) of 4
 dim ckey$ of 1
 dim error$ of 2
 close
 print chr$(14),chr$(8),
endproc begin
//
proc convert'file
 center("Convert Speedscript to Seq")
 repeat
  e:=0
  print
  center("Enter Speed text file name")
  print
  print "      [",
  get'name
  print "]"
  print
  inputname$:=name$
  if name$="" then goodbye
  pass "i0"
```

More ►

```
 open file 2,inputname$+",prg",read
 error$:=status$
 if error$<>"00" then
  e:=1
  print
  print " error$=",error$
  print
  if error$="62" then
   print
   print "file not found"
  endif
 endif
 close file 2
until e=0 // until file is
// opened properly
repeat
 e:=0
 center("Enter Seq file name")
 print
 print "      [",
 get'name
 print "]"
 print
 outputname$:=name$
 if name$="" then goodbye
 pass "i0"
 open file 3,outputname$,write
 error$:=status$
 if error$<>"00" then
  e:=1
  print
  print " error$=",error$
  if error$="63" then
   print
   print " file exists"
  endif
  close file 3
 endif
until e=0
print file 3: ""
open file 2,inputname$+",p,r",read
center("Converting the file")
sys 828 //run machine language
error$:=status$
if error$<>"00" then
 close
 print
 print " error$=",error$
 print
 center("Error in conversion")
 print chr$(9),
 end
endif
 print
 center("Closing the file")
 close
endproc convert'file
//
proc poke'machine'language
 restore
 total:=0
 for z:=828 to 947 do
  read y
  total:+y
  poke z,y
 endfor z
 if total<>15955 then
  print "ERROR in DATA statements!!!"
  goodbye
 endif
 data 162,2,32,198,255,160,0,32
 data 207,255,153,0,192,165,144,41
 data 64,208,6,200,208,241,169,0
 data 44,169,255,133,255,132,254,32
 data 204,255,160,0,185,0,192,32
 data 148,3,153,0,192,200,208,244
 data 162,3,32,201,255,160,0,185
 data 0,192,240,3,32,210,255,165
 data 255,208,10,200,208,241,32,204
 data 255,162,0,240,179,196,254,240
 data 4,200,208,227,0,76,204,255
 data 201,31,176,4,105,64,208,22
 data 201,31,208,4,169,13,208,14
 data 201,64,144,10,201,91,176,4
 data 105,128,208,2,169,0,96,0
endproc poke'machine'language
//
proc goodbye
 close
 print chr$(9),
 end
endproc goodbye
//
proc space(n)
 k:=int(n)
 for i:=1 to k do
  print " ",
 endfor i
endproc space
//
proc center(message$)
 space(20-len(message$)/2)
 print message$
endproc center
//
```

More ►

```
proc get'char
 ccursor$(0):="> "+chr$(157)+chr$(157)
 ccursor$(1):="< "+chr$(157)+chr$(157)
 n:=0; d:=false
 d:=0
 repeat
  print ccursor$(d),
  d:=not d
  repeat
   ckey$:=key$
   n:=n+1
   if n>30 then n:=0
  until n=0 or ckey$<>chr$(0)
 until ckey$<>chr$(0)
endproc get'char
//
proc get'name
 name$:=""
 repeat
  get'char
  s:=ord(ckey$)
  if ckey$=chr$(20) and len(name$)>0 th
  en //wrap line
   name$:=name$(1:len(name$)-1)
   print ckey$,
  endif
  if (s>31 and s<96) or (s>192 and s<22
  4) then //wrap line
   name$:=name$+ckey$
   print ckey$,
  endif
 until ckey$=chr$(13)
endproc get'name ■
```

2.0 Scrolling Text

An interesting procedure to move the screen down by using the getscreen / setscreen commands in the system package was sent to us from the Dutch COMAL Users Group. With slight modifications, we found that WINDOWS, or areas, of text can be scrolled on the screen. The first procedure scrolls the entire screen down one line and makes the top line blank. The second two procedures scroll just an area of text up or down. These procedures may not work at machine code speed, but they are fairly quick, and are readable.

More ►

```
PROC scroll'down CLOSED
  USE system
  DIM screen$ OF 1505, char$ OF 3
  getscreen(screen$)
  screen$(66:1505):=screen$(6:1465)
  back:=ORD(screen$(2))
  char$:="  "+CHR$(back+back*16)
  FOR row#:=6 TO 63 STEP 3 DO
    screen$(row#:row#+2):=char$
  ENDFOR row#
  setscreen(screen$)
ENDPROC scroll'down
//
PROC window'down(bottom,top) CLOSED
  USE system
  DIM screen$ OF 1505, char$ OF 3
  IF bottom>0 AND bottom<24 AND top>1 AN
  D top<26 AND bottom<top THEN // wrap
    b:=(bottom-1)*60; t:=(top-1)*60
    getscreen(screen$)
    screen$(66+b:t+65):=screen$(6+b:t)
    back:=ORD(screen$(2))
    char$:="  "+CHR$(back+back*16)
    FOR row#:=6+b TO 63+b STEP 3 DO
      screen$(row#:row#+2):=char$
    ENDFOR row#
    setscreen(screen$)
  ENDIF
ENDPROC window'down
//
PROC window'up(bottom,top) CLOSED
  USE system
  DIM screen$ OF 1505, char$ OF 3
  IF bottom>0 AND bottom<24 AND top>1 AN
  D top<26 AND bottom<top THEN // wrap
    b:=(bottom-1)*60; t:=(top-1)*60
    getscreen(screen$)
    screen$(6+b:t):=screen$(66+b:t+65)
    back:=ORD(screen$(2))
    char$:="  "+CHR$(back+back*16)
    FOR row#:=6+b TO 63+b STEP 3 DO
      screen$(row#:row#+2):=char$
    ENDFOR row#
    setscreen(screen$)
  ENDIF
ENDPROC window'up ■
```

Direct Reduction Repayment Schedule

by Gordon Shigley

COMAL listings for financial programs are almost a joy to read because of their clarity. This short program in COMAL 2.0 calculates the installment repayments of direct-reduction plans such as those associated with mortgages. Four procedures do all the work:

info
calculations
output'selection
printout

The first procedure takes in information on the principle being financed, the number of payments per year, the time over which payment is planned, and the interest rate--enter 12.8% as 12.8 or .128 (the program will figure it out)--on which the loan repayment is based.

```
PROC info
  PAGE
  PRINT
  PRINT " Cash value being financed",
  INPUT AT 0,0,7: ": $": principle
  PRINT " Number of payments per year",
  INPUT AT 0,0,3: ": ": n
  PRINT " Total number of payments",
  INPUT AT 0,0,3: ": ": t
  PRINT " Interest rate per year",
  INPUT AT 0,0,5: ": ": interest
  IF interest>1 THEN
    interest:=interest/100
  ENDIF
ENDPROC info
```

It makes use of the protected-field input that locates the row and column, with a limit on the length of any possible response. For example, INPUT AT 3,2,7 permits the user to give a reply beginning at the 3rd row and 2nd column. The cursor will move no more than 7 spaces for any reply. In the above examples INPUT AT 0,0,7 means to start the INPUT field where the cursor currently is, for a length of 7 spaces.

The CALCULATIONS procedure manipulates the values which the variables hold to calculate the payment, total payback, and interest charges. All can be found in any financial text. The first is the most involved. The [up arrow] in the second line refers to the up arrow key above the RETURN key on the right hand side of the keyboard. This symbol should be read "to the power of".

```
PROC calculations
  temp:=(1+(interest/n))
  temp:=(1-(1/temp[up arrow]t))
  payment:=principle*(interest/n)/temp
  payback:=payment*t
  interest'chg:=payback-principle
ENDPROC calculations
```

Next, the OUTPUT'SELECTION determines whether to send the output to the screen or to the printer and sets the variable that a later ZONE command will use to fit the output on a 40-column screen or an 80-column hard copy:

```
PROC output'selection
  REPEAT
    PRINT AT 7,2: "Output to screen";
    PRINT "or printer?";
    INPUT AT 0,0,1: "(s/p)":reply$
  UNTIL reply$ in "ps"
  IF reply$ = "p" THEN
    SELECT OUTPUT "lp:"
    column'spacing:=20
  ELSE
    SELECT OUTPUT "ds:"
    column'spacing:=10
  ENDIF
ENDPROC output'selection
```

More ►

The REPEAT..UNTIL combined with the PRINT AT means that the question will be repeated at exactly the same area of the screen until either of two valid letters are entered. The column'spacing variable is needed only if you wanted the output in columns twice as wide for the printer.

The procedure "printout" creates the table of payments using the PRINT USING and ZONE commands for output that is aligned within appropriately-spaced columns. Each payment number, account balance, interest charge, and payment amount will be displayed for the entire range of payments following the summary for total payback and monthly payment, and an underlined header for each of these columns. Notice that after printing a line of information, the payment number is incremented and the interest charge and principle are recalculated for the next line.

```
PROC printout
  PRINT "Amount borrowed:        ";
  PRINT USING "$#######.##": principle
  PRINT "Payback for this loan is";
  PRINT USING "$#######.##": payback
  PRINT "Each payment will be:   ";
  PRINT USING "   $####.##": payment
  ZONE column'spacing
  PRINT "PAYMENT","ACCOUNT",
  PRINT "INTEREST","PAYMENT"
  PRINT "NUMBER","BALANCE",
  PRINT "AMOUNT","AMOUNT"
  ZONE 0
  FOR i#:=1 TO (column'spacing*4) DO
    PRINT "=",
  ENDFOR i#
  PRINT
  ZONE column'spacing
  payment'number:=1
  interest'chg:=principle*interest/n
  WHILE principle >= 0.009 DO
    PRINT USING "###": payment'number,
    PRINT USING "$#####.##": principle,
    PRINT USING "$###.##": interest'chg,
    PRINT USING "$####.##": payment
    principle:+interest'chg-payment
    payment'number:+1
    interest'chg:=principle*interest/n
    IF principle+interest'chg<=payment
    THEN // wrap line
      payment:=principle+interest'chg
    ENDIF
  ENDWHILE
  SELECT OUTPUT "ds:"
ENDPROC printout
```

Notice the commas after the first three PRINT USING statements inside the WHILE loop. They respond to the ZONE command and space the columns either 10 or 20 columns apart.

Well, this gives some practice with common COMAL 2.0 commands and you have a program that produces a net accounting for each payment, the total principal paid off, and the total interest for the life of a direct-reduction loan (and all without a single GOTO statement). ■

Character Codes

The following is a list of some ASCII character codes which can be used in COMAL, along with the action provided:

CODE	MEANING
3	stop key
4	graphics dump
5	white characters
11	erase to end of line
12	go to end of line
13	return
17	cursor down
18	reverse
19	home
20	delete
28	red
29	cursor right
30	green
31	blue
129	orange
144	black
145	cursor up
146	reverse off
147	clear / PAGE
157	cursor left
158	yellow

Using the Interrupt Command — 2.0

By Jesse Knight

In COMAL TODAY 7, Tom Kuiper described some features he would like to see in COMAL. The 2.0 cartridge does have one of the features mentioned. It is interrupt handling. The problem is that UniComal hasn't told people how to use it. It is covered briefly, but not fully, in THE COMAL HANDBOOK under the command **INTERRUPT.**

If you had a procedure called handle'irq that you wanted to execute when a special COMAL interrupt occured, you would use the command:

```
INTERRUPT handle'irq
```

If you wanted to turn off the interrupt handler you would use **INTERRUPT** without a procedure name:

```
INTERRUPT
```

While a COMAL program is running, six bits in location EXCINF ($004D) are used as flags. Bits 0 and 6 are not used. As expected, if the bit is a one it means true or on and if it is a zero, it means false or off. Some of these flags are used in connection with the INTERRUPT command. I'll describe what each bit is used for.

Bits 1 and 3 relate to the STOP key. Bit 1 indicates the status of the TRAP ESC command. When bit 1 is set, indicating the STOP key is disabled, bit 3 indicates if the STOP key has been pressed. This is how the value of ESC is determined.

Bit 4 is used in connection with the INTERRUPT command. It is set when INTERRUPT <procedure name> is given and reset to zero when just INTERRUPT is used.

Bit 2 is a flag that controlls calls to the procedure specified by the INTERRUPT command. When INTERRUPT << procedure name >> is in effect, setting this bit will cause the specified procedure to be executed.

When the proper flags are set, causing the call to the procedure to be made, COMAL handles it as if the next instruction it encounters was a call to the procedure. For an example, look at the following code:

```
INTERRUPT hanlde'irq
//
FOR x=1 TO 20 DO
  PRINT x
ENDFOR X
//
PROC handle'irq
  PRINT "Call made to HANDLE'IRQ"
ENDPROC hanlde'irq
```

Assume that the program above was running and bit 2 of EXCINF was set when the variable 'x' equals 11. The result would be the same if the INTERRUPT command was removed and the statement **IF x=11 THEN handle'irq** was inserted between the PRINT and ENDFOR statements. Consider what would happen if the following code was used instead:

```
INTERRUPT handle'irq
//
work
//
PROC work CLOSED
  IMPORT handle'irq
  FOR x=1 to 20 DO
    PRINT x
  ENDFOR x
ENDPROC work
//
PROC handle'irq
  PRINT "Call made to HANDLE'IRQ"
ENDPROC handle'irq
```

More ►

With the same conditions as above, the results would be the same. Note the IMPORT handle'irq statement added in the closed procedure. This is a neccesity because of how the INTERRUPT command is handled. If the IMPORT statement was not used, and the call to the interrupt procedure was enabled while the closed procedure was executing, the result would be an 'unknown statement or procedure' error.

When using the INTERRUPT command in a program with closed procedures or functions, to be safe you should do one of two things. The first option is to import the procedure for handling the interrupts into ALL closed procedures or functions. The second option is to disable the interrupts (with INTERRUPT) before calling any closed procedures or function, then restore the interrupts (with INTERRUPT << procedure name >>) after returning.

Bit 5 of EXCINF controls calls to a machine language routine during the IRQs. The vector at $C7E2 (USRQVC) is used to point to the code. Normally it points to the subroutine DUMMY at $CA2F, which is just an RTS. Changing this vector and setting bit 5 of EXCINF results in the machine code being called during each IRQ. For code relating to the INTERRUPT command, this is simpler than using the normal IRQ vector.

Because of how the cartridge handles the IRQ interrupts, it is not possible to directly enter code in the RAM between $8000 and $BFFF. To call code located there, a small routine, located in protected memory outside of this area would have to be used. A good place would be in the unused bytes at $C86A-C87A. If the entire routine is short enough it could be placed there.

Here's a demo program:

```
PAGE
setup
TRAP ESC-
count:=0
//
REPEAT
  PRINT ""19"Press Q to quit"
  PRINT "count = ",count
  count:+1
UNTIL KEY$ in "qQ"
TRAP ESC+
//
PROC irq'proc
  temp:=ESC
  PRINT "stop key was pressed"
  PRINT "press space to continue"
  WHILE KEY$<>""0"" DO NULL
  WHILE KEY$<>" " DO NULL
  PAGE
ENDPROC irq'proc
//
PROC setup CLOSED
  FOR lp#:=0 TO 12 DO
    READ byte
    POKE $c86a+lp#,byte
  ENDFOR lp#
  POKE $c7e2,$6a
  POKE $c7e3,$c8
  POKE $4d,PEEK($4d) BITOR %00100000
  DATA $a5,$4d,$29,$08,$f0,$06,$a9
  DATA $04,$05,$4d,$85,$4d,$60
ENDPROC setup
```

The procedure SETUP pokes a short ML routine into the unused bytes discussed earlier. Then USRQVC is set to point to the code and bit 5 of EXCINF is set so the code will be called during the IRQ. All the ML does is test bit 3 of EXCINF, which gets set when the STOP key is pressed. When it is set, the ML sets bit 2 of EXCINF, causing a call to the interrupt procedure. To prevent the program from stopping when the STOP key is pressed, the **TRAP ESC-** command is used.

The main program loop increments a variable and displays it at the top of the screen. The program can be stopped by pressing 'q'. When the STOP key is pressed the procedure IRQ'PROC is called. It displays a message and waits for the SPACE bar to be pressed. ■

User Groups

OCTOBER is our User Group Appreciation Month. We encourage you to include any of our COMAL disks in your Group's disk library. COMAL TODAY is designed to help your members understand COMAL. Send in the USER GROUP registration card to get a free COMAL TODAY subscription for your President, Newsletter Editor, or COMAL SIG Leader. Plus until October 31, 1985 your members can subscribe to COMAL TODAY for only $13 each. If 13 or more subscribe **at one time** we'll send your group a FREE COMAL 2.0 Cartridge.

[ ] 13 people are subscribing (payment enclosed). Send us a FREE COMAL Cart.
[ ] Less than 13 subscribers - payment enclosed at $13 each.
>>>>> **RENEWALS OK TOO - MUST INCLUDE THEIR SUBSCRIBER NUMBER**

GROUP:_____

Name	Address	City	State	ZIP
1.				
2.				
3.				
4.				
5.				
6.				
7.				
8.				
9.				
10.				
11.				
12.				
13.				

Making Stars in 2.0

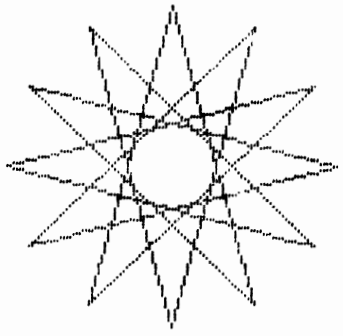
by Chris Zamara

The following COMAL 2.0 program demonstrates graphics commands using Splitscreen mode to INPUT data.

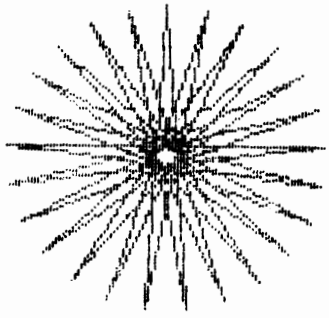
```
// delete "star-zamara"
// save   "star-zamara"
//  by chris zamara, may 26/85
//
// draw an n pointed star
USE graphics
USE turtle
splitscreen
PAGE
size:=100
LOOP
 INPUT ""19"number of points? ": points
 clear
 xstart:=INT(160-size/2)
 ystart:=INT(100-size/2)
 moveto(xstart,ystart)
 pendown
 star(size,points)
ENDLOOP

PROC star(size,points)
 // ** draw an N-pointed star **
 // first  calculate the angle to
 // turn at each point
 CASE (points MOD 4) OF
 WHEN 0
   angvar:=points
 WHEN 2
   angvar:=points/2
 OTHERWISE
   angvar:=points*2
 ENDCASE
 angle:=180-360/angvar
 // now draw the star
 setheading((180-angle)/2)
 FOR i#:=1 TO points DO
  forward(size)
  right(angle)
 ENDFOR i#
ENDPROC star ■
```

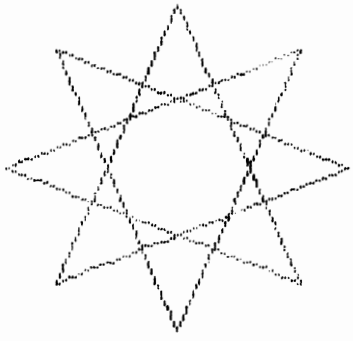
number of points? 12



number of points? 25



number of points? 8



Custom Error Messages

Would you like to customize COMAL's error messages? Make some silly messages to cure the programming blues. Or change the wording on the ones that bother you. Foreign language messages also can be created using this system. A program from the Holland COMAL Users Group is included on TODAY DISK #8 to do this - just change the DATA STATEMENTS. The program reads the error messages from DATA STATEMENTS and creates a package ready for use with COMAL 2.0. If you create a general use message package, we hope you will submit it for use on a future TODAY DISK.

```
// delete "make'err'english"
// save   "make'err'english"
//  by The Dutch User Group
//
DIM code$ OF $2000
RESTORE text
REPEAT
  READ number
  IF number=$ff THEN
    code$:+CHR$($ff)
  ELSE
    code$:+CHR$(number)
    READ et$
    code$:+CHR$(LEN(et$))
    code$:+et$
  ENDIF
UNTIL number=255
//
index:=LEN(code$)
//
REPEAT
  READ et$
  code$:+CHR$(LEN(et$))
  code$:+et$
UNTIL EOD
//
RESTORE
PAGE
PRINT "Make English error file package."
PRINT
INPUT "Package name: ": name$
PRINT "working..."
ln:=LEN(name$)
l:=LEN(code$)
e:=29+l+ln+$a000
tabel:=11+ln+$a000-1
init:=tabel+1
texten:=28+ln+$a000-1
second:=texten+index+2
scode$:=CHR$($76)+CHR$(e MOD 256)
scode$:+CHR$(e DIV 256)
scode$:+CHR$($2f)+CHR$($ca)
scode$:+CHR$(ln)+name$
scode$:+CHR$(tabel MOD 256)
scode$:+CHR$(tabel DIV 256)
scode$:+CHR$(init MOD 256)
scode$:+CHR$(init DIV 256)
scode$:+CHR$(0)+CHR$($a2)
scode$:+CHR$(texten MOD 256)
scode$:+CHR$($a9)
scode$:+CHR$(texten DIV 256)
FOR t:=1 TO 12 DO
  READ c
  scode$:+CHR$(c)
ENDFOR t
scode$:+CHR$(second MOD 256)
scode$:+CHR$(second DIV 256)
code$:=scode$+code$
//
OPEN FILE 1,name$,WRITE
PRINT FILE 1: CHR$(0)+CHR$($a0),
PRINT FILE 1: code$,
CLOSE
//
DATA $a0,$76,$8e,$e4,$c7,$8d
DATA $e5,$c7,$8c,$e6,$c7,$60
//
text:
DATA 0,"report error"
DATA 1,"argument error"
DATA 2,"overflow"
DATA 3,"division by zero"
DATA 4,"substring error"
DATA 5,"value out of range"
DATA 6,"step = 0"
DATA 7,"illegal bound"
DATA 8,"error in print using"
DATA 10,"index out of range"
```

More ►

```
DATA 11,"invalid file name"
DATA 13,"verify error"
DATA 14,"program too big"
DATA 15,"bad comal code"
DATA 16,"not comal program file"
DATA 17,"program for other comal versio
n" //wrap line
DATA 18,"unknown file attribute"
DATA 30,"invalid color"
DATA 31,"invalid boundary"
DATA 32,"invalid shape number"
DATA 33,"shape length must be 64"
DATA 34,"invalid sprite number"
DATA 35,"invalid voice"
DATA 36,"invalid note"
DATA 51,"system error"
DATA 52,"out of memory"
DATA 53,"wrong dimension in parameter"
DATA 54,"parameter must be an array"
DATA 55,"too few indices"
DATA 56,"cannot assign variable"
DATA 57,"not implemented"
DATA 58,"con not possible"
DATA 59,"program has been modified"
DATA 60,"too many indices"
DATA 61,"function value not returned"
DATA 62,"not a variable"
DATA 67,"parameter lists differ or not
closed" //wrap line
DATA 68,"no closed proc/func in file"
DATA 69,"too few parameters"
DATA 70,"wrong index type"
DATA 71,"parameter must be a variable"
DATA 72,"wrong parameter type"
DATA 73,"non-ram load"
DATA 74,"checksum error in object file"
DATA 75,"memory area is protected"
DATA 76,"too many libraries"
DATA 77,"not an object file"
DATA 78,"no matching when"
DATA 79,"too many parameters"
DATA 101,"syntax error"
DATA 102,"wrong type"
DATA 103,"statement too long or too com
plicated" //wrap line
DATA 104,"statement only, not command"
DATA 106,"line number range: 1 to 9999"
DATA 108,"procedure/function does not e
xist" //wrap line
DATA 109,"structured statement not allo
wed here" //wrap line
DATA 110,"not a statement"
DATA 111,"line numbers will exceed 9999"
DATA 112,"source protected!!!"
DATA 113,"illegal character"
DATA 114,"error in constant"
DATA 115,"error in exponent"
DATA 200,"end of data"
DATA 201,"end of file"
DATA 202,"file already open"
DATA 203,"file not open"
DATA 204,"not input file"
DATA 205,"not output file"
DATA 206,"numeric constant expected"
DATA 207,"not random access file"
DATA 208,"device not present"
DATA 209,"too many files open"
DATA 210,"read error"
DATA 211,"write error"
DATA 212,"short block on tape"
DATA 213,"long block on tape"
DATA 214,"checksum error on tape"
DATA 215,"end of tape"
DATA 216,"file not found"
DATA 217,"unknown device"
DATA 218,"illegal operation"
DATA 219,"i/o break"
DATA 230,""13"end at "
DATA 231,""13"stop at "
DATA 232," bytes free."13""13""
DATA 233,"error"
DATA 242,"at "
DATA 243,"prog  data  free"13""
DATA 244,"the program stopped at"
DATA 245,"which is called at"
DATA 246,"within"
DATA 255
DATA ""249""
DATA ""250""
DATA ""97" or "95""
DATA "unknown label"
DATA "name already defined"
DATA "not a label"
DATA "string not dimensioned"
DATA "not a package"
DATA "expression"
DATA "variable"
DATA "operand"
DATA "variable name"
DATA "num. variable name"
DATA "numeric expression"
DATA "string expression"
DATA "F or "96""
DATA ""97" or J"
```

More ►

```
DATA "F or G"
DATA "K or M"
DATA "mode"
DATA "constant"
DATA "line number"
DATA "binary constant"
DATA "label"
DATA ""113"/"114"/"117""
DATA ""117""
DATA ""118""
DATA ""240""
DATA ""245""
DATA ""176""
DATA ""119""
DATA ""178"/"147"/"116""
DATA ""178"/"147"/"116""
DATA ""116""
DATA " not expected"
DATA " missing"
DATA " expected, not "
DATA "real constant"
DATA "integer constant"
DATA "string constant"
DATA "name"
DATA "integer name"
DATA "string name"
DATA " not allowed in control structure
s" //wrap line
DATA " without "
DATA ""244""
DATA ""178""
DATA ""226" allowed in closed proc/func
 only" //wrap line
DATA "wrong type of "
DATA "wrong name in "
DATA ""104" or N"
DATA "hex constant"
DATA "illegal "128""
DATA "unknown statement or procedure"
DATA "not a procedure"
DATA "unknown variable"
DATA "wrong type"
DATA "wrong function type"
DATA "not an array nor a function"
DATA "not a simple variable"
DATA "unknown array or function"
DATA "wrong array type"
DATA "import error"
DATA "unknown package" ■
```

Fast Directory 2.0

The following procedure was adapted from the FAST DIRECTORY program by George Jones that was printed in COMAL Today 7. Individual file entries are in 32 byte strings. The first byte tells what type of file it is and if it was properly closed. The second and third bytes are the starting track and sector of the file. The fourth byte through the 19th contain the file name padded with shifted spaces (chr$(160)). The rest of the entry contains other information not normally needed.

```
PROC directory(REF disk$,REF id$,REF co
unt,REF bam$,REF entry$()) CLOSED //wrap
  //
  count:=0; sector#:=1
  MOUNT
  OPEN FILE 2,"u8:#2/s2/t+/d-"
  setblock(0,4)
  bam$:=GET$(2,140)
  disk$:=GET$(2,18); id$:=GET$(2,2)
  REPEAT
    setblock(sector#,0)
    get'entries
  UNTIL sector#>18
  CLOSE FILE 2
  //
  PROC get'entries
    count:+1
    track#:=ORD(GET$(2,1))
    sector#:=ORD(GET$(2,1))
    entry$(count):=GET$(2,32)
    FOR i#:=2 TO 8 DO
      IF entry$(count)(4)<>""0"" THEN c
      ount:+1 //wrap line
      entry$(count):=GET$(2,32)
    ENDFOR i#
    // check for NULL file
    IF entry$(count)(4)=""0"" THEN coun
    t:-1 //wrap line
  ENDPROC get'entries
  //
  PROC setblock(sec,postion)
    PASS "u1: 2 0 18 "+STR$(sec)
    PASS "b-p: 2 "+STR$(postion)
  ENDPROC setblock
ENDPROC directory ■
```

Word Game for 2.0

by Daniel Horowitz

[Editors Note: This is a professional quality game. The whole program is listed for your information. We don't expect you to type in a program this large. It is on TODAY DISK 8.]

This program is similar to the "fill in the missing letters" games that you find in the newspapers. Only this time you can make up your own words, and select which letters you want hidden. Text can be up to eight lines.

WORD GAME clearly falls into the category of educational software. I use it in teaching English to foreign students before they enter Western Illinois University, but it is really ideal for teaching reading and/or vocabulary to anyone, especially children. Texts are easy to enter, so parents or teachers can create individualized lessons with content that is familiar and interesting to the learner. Although it seems like a "guessing" game, the guesses made by a player are based on his or her predictive abilities, which are the basis for understanding written text. Anyone interested in a very interesing book on the relationship between inference making and reading should take a look at Roger Schank's **Reading and Understanding: Teaching Reading from the Perspective of Artificial Intelligence.**

I would be interested in hearing from other language teachers who use or write programs in their work. My address is: Daniel Horowitz, WESL Institute, Western Illinois University, Macomb, IL 61455.

Sample Text:

```
   C-M-L -s --sy t- l--rn -nd p-w-rf-l
t- -s-. Th-s- wh- w-nt t- g- -n t-
-v-n m-r- pr-f-ss--n-l h-gh l-v-l
l-ng--g-s, s-ch -s P-sc-l -r -d-, f-nd
-t n-t-r-l t- d- s-.
```

```
DIM ans$ OF 1, lines$(8) OF 38
DIM name$ OF 16, punc$ OF 40
DIM all'names$(40) OF 16
DIM heading$ OF 13, test1$ OF 9
DIM char$ OF 1, text$ OF 304
DIM alf$ OF 52, guess'text$ OF 304
test1$:="123456789"
punc$:="!#$%&'()+-  *@:;[]=?/><.,"
punc$:+"1234567890"
alf$:="abcdefghijklmnopqrstuvwxyz"
alf$:+"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
USE system
textcolors(14,14,0)
ZONE 0
beginning
//
PROC writer'lines
  create'it
  this'game'name
  writing'files
  beginning
ENDPROC writer'lines
//
PROC player'lines
  play'instructions
  menu
  text'create
  questions
  show'it
  play'lines
  beginning
ENDPROC player'lines
//
PROC accept(prompt$,valid$,REF reply$,ma
x) CLOSED //wrap line
  IMPORT fetch
  PRINT prompt$,
  PRINT "['",
  FOR x:=2 TO max DO PRINT " ",
  PRINT "]",
  FOR x:=1 TO max+1 DO PRINT ""157"",
  fetch(reply$,valid$,max)
  PRINT
ENDPROC accept
//
PROC get'valid(REF c$,valid$) CLOSED
  REPEAT
    c$:=KEY$
  UNTIL c$ IN valid$
ENDPROC get'valid
```

More ►

```
PROC beginning
  LOOP
    PAGE
    PRINT AT 8,5: "Do you want to..."
    PRINT AT 10,9: "1. Play a game?"
    PRINT AT 12,9: "2. Create a game?"
    PRINT AT 14,9: "3. Quit?"
    PRINT
    accept("Your choice?","123",ans$,1)
    CASE ans$ OF
    WHEN "1","P","p"
      player'lines
    WHEN "2","C","c"
      writer'lines
    WHEN "3","Q","q"
      END "Thank you for playing."
    OTHERWISE
      NULL
    ENDCASE
  ENDLOOP
ENDPROC beginning
//
PROC create'it
  PAGE
  accept("Instructions(y,n)?","yn",ans$,
  1) //wrap line
  IF ans$="y" THEN instructions
  n:=1
  PRINT AT 3,3: "Enter your text:"
  REPEAT
    accept("","b",lines$(n),38)
    n:=n+1
  UNTIL n=9 OR lines$(n-1)=""
  REPEAT
    PAGE
    CURSOR 3,1
    FOR n:=1 TO 8 DO
      CURSOR 2*n+1,1
      PRINT n,">",lines$(n)
    ENDFOR n
    PRINT "Enter line number you want"
    INPUT "to change(0=all is OK): ": x
    IF x>0 AND x<9 THEN
      PRINT " Enter the changed line:"
      accept("","b",lines$(x),38)
    ENDIF
  UNTIL x=0
ENDPROC create'it
//
PROC key'press
  PRINT
  PRINT "Press any key to continue"
  WHILE KEY$<>""0"" DO NULL
  WHILE KEY$=CHR$(0) DO NULL
ENDPROC key'press
```

```
PROC fetch(REF a$,v$,max) CLOSED
  IMPORT get'valid
  DIM valid$ OF 80, b$ OF 1, also'valid$
   OF 25 //wrap line
  also'valid$:="!#$%&'()+@@*@:;[]=<>?/"
  a$:=""
  CASE v$ OF
  WHEN "a"
    valid$:="abcdefghijklmnopqrstuvwxyz
    ABCDEFGHIJKLMNOPQRSTUVWXYZ,. "//wrap
  WHEN "d"
    valid$:="0123456789"
  WHEN "b"
    valid$:="abcdefghijklmnopqrstuvwxyz
    0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ
    ,." //wrap line - all 3 on 1 line
  OTHERWISE
    valid$:=v$
  ENDCASE
  done:=FALSE; num:=0
  REPEAT
    get'valid(b$,valid$+CHR$(13)+CHR$(20
    )+also'valid$) //wrap line
    CASE b$ OF
    WHEN CHR$(13)
      done:=TRUE
    WHEN CHR$(20)
      IF num THEN
        num:-1
        IF num=max-1 THEN
          PRINT ""157"'"157"",
        ELSE
          PRINT " "157""157"'"157"",
        ENDIF
        a$:=a$(1:num)
      ENDIF
    OTHERWISE
      IF num<max THEN
        a$:=a$+b$
        num:+1
        PRINT b$;""157"'"157"",
        IF num=max THEN PRINT "]"157"",
      ENDIF
    ENDCASE
  UNTIL done
ENDPROC fetch
//
PROC questions
  PAGE
  PRINT AT 2,4: "How many wrong guesses
  before the" //wrap line
  CURSOR 4,5
  accept("answer is given(1-5)? ","12345
  ",ans$,1) //wrap line
```

More ►

```
  enough:=VAL(ans$)
  PRINT
  PRINT "   How do you want the text?"
  PRINT
  PRINT "1. H-- d- y-- w--- t-- t---?"
  PRINT
  PRINT "2. --w -o --u ---t --e ---t?"
  PRINT
  PRINT "3. H-w d- y-- w-nt th- t-xt?"
  PRINT
  PRINT "4. -o- -o -ou -a-- --e -e--?"
  PRINT
  PRINT "5. How -- you ---- the ----?"
  PRINT
  PRINT "6. H-w -o -o- w-n- t-e -e-t?"
  PRINT
  PRINT "7. --- -- --- ---- --- ----?"
  CURSOR 23,4
  accept("Your choice(1-7)> ","1234567",
  ans$,1) //wrap line
  initialize(ans$)
ENDPROC questions
//
PROC menu
  PAGE
  TRAP
    OPEN FILE 4,"dat.game'names",READ
  HANDLER
    CLOSE FILE 4
    PAGE
    PRINT "ERROR! No games on this disk"
    PRINT "Please make sure the file"
    PRINT """dat.game'names"" is on the"
    END "disk before you run it again."
  ENDTRAP
  READ FILE 4: heading$
  PRINT heading$
  n:=1
  WHILE NOT EOF(4) DO
    READ FILE 4: all'names$(n)
    IF n>20 THEN CURSOR n-20,21
    PRINT n,". ",all'names$(n)
    n:=n+1
  ENDWHILE
  CLOSE FILE 4
  REPEAT
    choose:=getnum(24,6,"Your choice(0=q
    uit)? ") //wrap line
  UNTIL choose>-1 AND choose<n
  IF choose=0 THEN
    beginning
  ELSE
    load'text(all'names$(choose))
  ENDIF
ENDPROC menu
```

```
PROC initialize(choice$)
  PAGE
  PRINT AT 9,9: "one moment, please..."
  guess'text$:=text$
  CASE choice$ OF
  WHEN "1"
    FOR n:=1 TO LEN(text$)-1 DO
      IF text$(n)=" " THEN
        guess'text$(n+1):=text$(n+1)
      ELIF text$(n) IN punc$ THEN
        guess'text$(n):=text$(n)
      ELSE
        guess'text$(n+1):="-"
      ENDIF
    ENDFOR n
    FOR n:=1 TO LEN(text$) DO
      IF text$(n)=" " THEN
        guess'text$(n):=" "
      ELIF text$(n)="'" AND THEN text$(n
      +1) IN alf$ THEN //wrap line
        guess'text$(n+1):="-"
      ENDIF
    ENDFOR n
    n:=1
    REPEAT
      guess'text$(n):=text$(n)
      n:=n+38
    UNTIL n>=LEN(text$)
  WHEN "2"
    FOR n:=2 TO LEN(text$) DO
      IF text$(n)=" " OR text$(n)=")" TH
      EN //wrap line
        guess'text$(n-1):=text$(n-1)
        IF text$(n-1) IN punc$ THEN gues
        s'text$(n-2):=text$(n-2) //wrap
      ELSE
        guess'text$(n):="-"
      ENDIF
    ENDFOR n
    n:=38
    REPEAT
      guess'text$(n):=text$(n)
      n:=n+38
    UNTIL n>=LEN(text$)
  WHEN "3"
    FOR n:=1 TO LEN(text$) DO
      CASE text$(n) OF
      WHEN "a","A","e","E","i","I","o","
      O","u","U" //wrap line
        guess'text$(n):="-"
      OTHERWISE
        guess'text$(n):=text$(n)
      ENDCASE
    ENDFOR n
```

More ►

```
WHEN "4"
  FOR n:=1 TO LEN(text$) DO
    CASE text$(n) OF
    WHEN "a","A","e","E","i","I","o","
    O","u","U" //wrap line
      guess'text$(n):=text$(n)
    OTHERWISE
      guess'text$(n):="-"
    ENDCASE
  ENDFOR n
WHEN "5"
  FOR n:=1 TO LEN(text$) DO
    IF text$(n) IN punc$ OR text$(n)="
     " THEN //wrap line
      guess'text$(n):=text$(n)
    ELSE
      guess'text$(n):="-"
    ENDIF
  ENDFOR n
  n:=1
  REPEAT
    n:=n+1
  UNTIL text$(n) IN alf$
  x:=n; y:=38
  REPEAT
    WHILE x<=y AND THEN guess'text$(x)
     IN punc$ DO //wrap line
      guess'text$(x):=text$(x)
      IF x<y THEN x:=x+1
    ENDWHILE
    IF x=y THEN x:=x+1; y:=y+38
    IF x<LEN(text$) AND x MOD 38<>1 TH
    EN //wrap line
      REPEAT
        x:=x+1
      UNTIL x=y OR text$(x) IN alf$
    ENDIF
    IF x=y THEN x:=x+1; y:=y+38
    IF x<LEN(text$) THEN
      REPEAT
        x:=x+1
      UNTIL x=y OR text$(x)=" "
    ENDIF
    IF x=y THEN x:=x+1; y:=y+38
    IF x<LEN(text$) AND x MOD 38<>1 TH
    EN //wrap line
      REPEAT
        x:=x+1
      UNTIL x=y OR text$(x) IN alf$
    ENDIF
    IF x=y THEN x:=x+1; y:=y+38
  UNTIL x>LEN(text$)
```

```
  WHEN "6"
    dash:=FALSE
    FOR n:=1 TO LEN(text$) DO
      IF text$(n) IN alf$ AND dash=FALSE
       THEN //wrap line
        guess'text$(n):=text$(n)
        dash:=TRUE
      ELIF text$(n) IN alf$ AND dash=TRU
      E THEN //wrap line
        guess'text$(n):="-"
        dash:=FALSE
      ENDIF
    ENDFOR n
  WHEN "7"
    FOR n:=1 TO LEN(text$) DO
      IF text$(n) IN alf$ THEN
        guess'text$(n):="-"
      ENDIF
    ENDFOR n
  ENDCASE
  FOR n:=1 TO LEN(text$) DO
    IF text$(n) IN punc$ THEN
      guess'text$(n):=text$(n)
    ELIF text$(n)=" " THEN
      guess'text$(n):=" "
    ENDIF
  ENDFOR n
ENDPROC initialize
//
FUNC file'exists(file'name$) CLOSED
  TRAP
    OPEN FILE 78,file'name$,READ
    CLOSE FILE 78
    RETURN TRUE
  HANDLER
    CLOSE FILE 78
    RETURN FALSE
  ENDTRAP
ENDFUNC file'exists
//
PROC this'game'name
  flag:=FALSE
  PAGE
  REPEAT
    CURSOR 8,2
    accept("Name of this game","b",name$
    ,16) //wrap line
    IF file'exists(name$) THEN
      PRINT
      PRINT "That name is already used"
    ELSE
      flag:=TRUE
    ENDIF
  UNTIL flag=TRUE
ENDPROC this'game'name
```

More ►

```
PROC play'lines
  USE system
  CURSOR 3,2
  rights:=0; wrongs:=0; wrong'guesses:=0
  ; z:=1 //wrap line
  find'dash(z,1)
  REPEAT
    char$:=inkey$
    CASE char$ OF
    WHEN CHR$(29)
      IF z=LEN(text$) THEN z:=1
      find'dash(z,1)
      wrong'guesses:=0
    WHEN CHR$(157)
      IF z=1 THEN z:=LEN(text$)
      find'dash(z,2)
      wrong'guesses:=0
    WHEN CHR$(17)
      z:=z+36
      IF z>=LEN(text$) THEN z:=18
      find'dash(z,1)
      wrong'guesses:=0
    WHEN CHR$(145)
      z:=z-36
      IF z<=0 THEN z:=LEN(text$)-20
      find'dash(z,2)
      wrong'guesses:=0
    WHEN text$(z)
      rights:=rights+1
      wrong'guesses:=0
      guess'text$(z):=text$(z)
      PRINT char$
      IF guess'text$<>text$ THEN find'da
      sh(z,1) //wrap line
    OTHERWISE
      wrong'guesses:=wrong'guesses+1; wr
      ongs:=wrongs+1 //wrap line
      IF wrong'guesses=enough THEN
        wrong'guesses:=0
        guess'text$(z):=text$(z)
        PRINT text$(z)
        IF guess'text$<>text$ THEN find'
        dash(z,1) //wrap line
      ENDIF
    ENDCASE
  UNTIL guess'text$=text$
  scores
ENDPROC play'lines
//
PROC writing'files
  TRAP
    OPEN FILE 3,"dat.game'names",WRITE
    WRITE FILE 3: "Your choices:"
  HANDLER
    OPEN FILE 3,"dat.game'names",APPEND
  ENDTRAP
  WRITE FILE 3: name$
  CLOSE FILE 3
  OPEN FILE 4,name$,WRITE
  FOR n:=1 TO 8 DO
    WRITE FILE 4: lines$(n)
  ENDFOR n
  CLOSE FILE 4
ENDPROC writing'files
//
PROC load'text(name$)
  OPEN FILE 3,name$,READ
  n:=1
  REPEAT
    READ FILE 3: lines$(n)
    n:=n+1
  UNTIL n=9 OR lines$(n-1)=""
  CLOSE FILE 3
ENDPROC load'text
//
PROC text'create
  n:=1
  WHILE n<9 AND THEN lines$(n)<>"" DO
    IF LEN(lines$(n))<38 THEN
      REPEAT
        lines$(n):=lines$(n)+CHR$(32)
      UNTIL LEN(lines$(n))=38
    ENDIF
    n:=n+1
  ENDWHILE
  n:=1; text$:=""
  REPEAT
    text$:=text$+lines$(n)
    n:=n+1
  UNTIL n=9 OR lines$(n-1)=""
ENDPROC text'create
//
PROC find'dash(REF z,r)
  IF r=1 THEN
    REPEAT
      z:=z+1
      IF z=LEN(text$) THEN z:=2
    UNTIL guess'text$(z)="-" OR z=LEN(gu
    ess'text$) //wrap line
    t:=1
    put'it'there(z)
  ELSE
    REPEAT
      z:=z-1
      IF z=1 THEN z:=LEN(text$)
    UNTIL guess'text$(z)="-" OR z=1
    k:=LEN(text$)
    put'it'there(z)
  ENDIF
ENDPROC find'dash
```

More ►

```
FUNC getnum(lin,pos,prompt$) CLOSED
  DIM num$ OF 9
  LOOP
    INPUT AT lin,pos: prompt$: num$
    TRAP
      RETURN VAL(num$)
    HANDLER
      NULL
    ENDTRAP
  ENDLOOP
ENDFUNC getnum
//
PROC show'it
  PAGE
  x:=1; y:=38; n:=3
  WHILE y<=LEN(guess'text$) DO
    PRINT AT n,2: guess'text$(x:y)
    n:=n+2; x:=y+1; y:=y+38
  ENDWHILE
  PRINT AT n,2: guess'text$(x:LEN(guess'
  text$)) //wrap line
ENDPROC show'it
//
PROC scores
  PRINT AT 19,2: "You made";rights;"corr
  ect guesses" //wrap line
  PRINT "and";wrongs;"wrong ones."
  IF wrongs<>0 THEN
    c:=rights+wrongs
    PRINT INT((rights/c)*100),"% of your
     guesses were correct" //wrap line
  ELSE
    PRINT "100% were correct!"
  ENDIF
  key'press
ENDPROC scores
//
PROC put'it'there(z)
  r:=((z-1) DIV 38*2)+3; c:=(z MOD 38)+1
  IF c=1 THEN c:=39
  CURSOR r,c
ENDPROC put'it'there
//
PROC instructions
  PAGE
  PRINT "1. Text can be up to 8 lines."
  PRINT "2. Press RETURN to enter line."
  PRINT "3. If text is less than 8"
  PRINT "   lines, enter an empty line"
  PRINT "   when you are finished."
  PRINT "4. Don't use quotation marks"
  PRINT "   or split words with dashes."
  PRINT "   There may be some empty"
  PRINT "   spaces at the end of lines."
  PRINT "5. You may use apostrophes(')"
  PRINT "   ONLY AS APOSTROPHES, NOT AS"
  PRINT "   SUBSTITUTES FOR QUOTE MARKS."
  PRINT "6. Leave at least one empty"
  PRINT "   space at the beginning and"
  PRINT "   end of the text."
  PRINT "7. After you enter the text,"
  PRINT "   you may revise it before"
  PRINT "   it goes onto the disk."
  key'press
ENDPROC instructions
//
PROC play'instructions
  PAGE
  CURSOR 11,5
  INPUT "Do you need instructions (y,n)?
  ": ans$ //wrap line
  IF ans$ IN "Yy" THEN
    PAGE
    PRINT AT 2,1: "1. In this game, you
    must guess letters" //wrap line
    PRINT "    which are missing from a
    text." //wrap line
    PRINT "2. If you guess correctly, th
    e letter" //wrap line
    PRINT "    appears. After a number"
    PRINT "    of wrong guesses, the com
    puter will" //wrap line
    PRINT "    give you the correct lett
    er." //wrap line
    PRINT "3. You can guess at any lette
    r at any" //wrap line
    PRINT "    time by moving the cursor
     to it." //wrap line
    PRINT "4. You begin by selecting a g
    ame from" //wrap line
    PRINT "    the menu."
    PRINT "5. You then tell the computer
    how" //wrap line
    PRINT "    many wrong answers it sho
    uld accept" //wrap line
    PRINT "    before giving you the ans
    wer." //wrap line
    PRINT "6. Finally, there are seven d
    ifferent" //wrap line
    PRINT "    ways you can choose to se
    t up the" //wrap line
    PRINT "    text. Clearly, some are h
    arder than" //wrap line
    PRINT "    others."
    PRINT "7. Have fun!"
    key'press
  ENDIF
ENDPROC play'instructions
```

Calling BASIC Programs

(for experts only)

The following COMAL 2.0 package was written in Holland. It is listed in the Commodore Editor/Assembler format, but could be converted for other assemblers. The file "comsymb" is not needed to assemble this source code, because all symbols used are defined at the beginning. This package will switch the computer to BASIC, load a BASIC program, RUN the program, and when finished, return to COMAL. You could use this package in a master menu program to call other programs.

NOTE: When your BASIC program ends the computer will automatically go back into COMAL mode, but any previous COMAL program in memory will be lost.

The name of this package is 'BASIC'. In order to use the package you must LINK it to your program. This is done with the command:

```
LINK "pkg.basic"
```

This will load the package from disk and link it onto your program. Once linked to the program in memory, the package will be SAVEd and LOADed with the program. To use the GO command within the package you must first issue the command:

```
USE basic
```

Then to LOAD and RUN a BASIC program you use the command:

```
GO(<filename>)
```

The <filename> can be a string variable or a string constant. The following program example will show you how to use this package. (Remember, you must LINK the package to the program before you RUN it.)

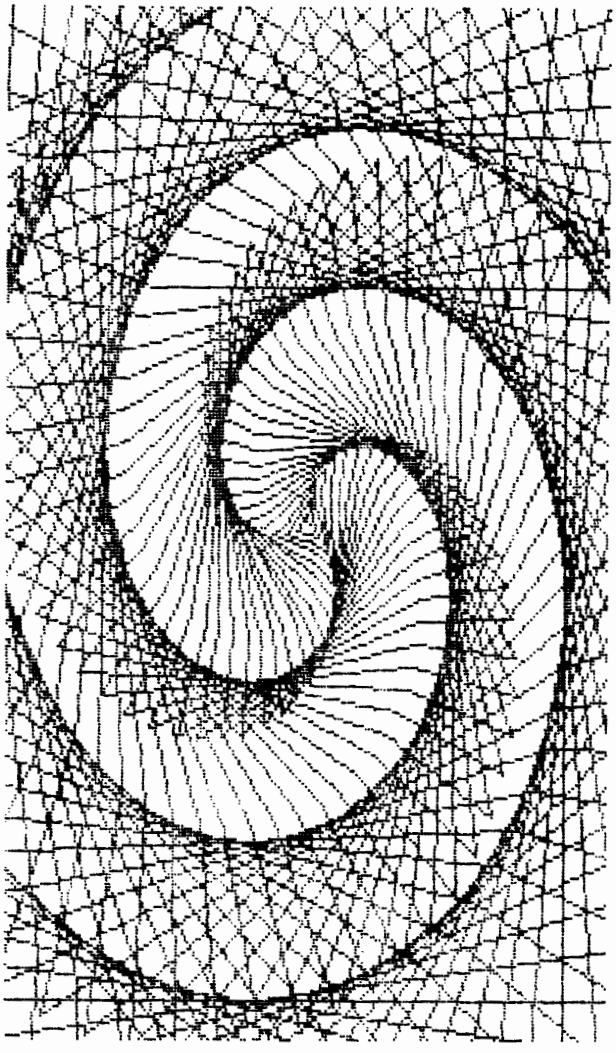
```
PAGE
USE basic
INPUT "BASIC program name: ": name$
PRINT
PRINT "Going to";name$
PRINT "Please wait..."
go(name$)
```

The source code for this package is listed here, and is on the back side of TODAY DISK 8, both source code and ready to LINK form. As you can see, packages can expand COMAL, even into using other languages.

```
        * =  $8009
setnam  =  $ffbd
setlfs  =  $ffba
load    =  $ffd5
open    =  $ffc0
chrout  =  $ffd2
proc    =  112
endpr   =  126
value   =  114
str     =  2
        .byte %01000110
        .word eind+17
        .word $ca2f
        .byte 5,'basic' ;package name
        .word procs
        .word $ca2f
        .byte 0
procs   .byte 2,'go'    ;procedure name
        .word head
        .byte 0
head    .byte proc,lo,hi,1
        .byte value+str,endpr
adres   =  *
lo      =  <adres
hi      =  >adres
        lda  #$01
        jsr  $c896
        ldy  #$02
        lda  ($45),y
        bne  argerr
        iny
```

More ►

```
        lda  ($45),y
        beq  argerr
        cmp  #$11
        bcc  ok
argerr  ldx  #$01
        jmp  $c9fb
ok      sta  eind
        ldx  #$00
        iny
name    lda  ($45),y
        sta  eind+1,x
        inx
        iny
        cpx  eind
        bcc  name
        sei
        lda  #$37
        sta  $01
        ldx  #$e0
        ldy  #$07
        lda  #$00
port    sta  $df00,y
        dey
        bpl  port
        stx  $de00
        stx  $8008
        ldx  #$ff
        txs
        cld
        jsr  $fda3
        jsr  $fd50
        jsr  $fd15
        jsr  $ff5b
        cli
        jsr  $e453
        jsr  $e3bf
        jsr  $e422
        ldx  #$fb
        txs
        lda  #<nmi
        ldy  #>nmi
        sta  $0318
        sty  $0319
        lda  #<back
        ldy  #>back
        sta  $0302
        sty  $0303
        lda  #$80
        sta  $9d
        lda  eind
        ldx  #<na
        ldy  #>na
        jsr  setnam
        ldx  #$08
        ldy  #$ff
        jsr  setlfs
        lda  #$00
        ldx  $2b
        ldy  $2c
        jsr  load
        bcc  noerr
        jmp  $e0f9
noerr   lda  $90
        and  #$bf
        beq  nolerr
        jmp  $e19c
nolerr  stx  $2d
        sty  $2e
        lda  #$02
        ldx  #<com
        ldy  #>com
        jsr  setnam
        lda  #$01
        ldx  #$08
        ldy  #$6f
        jsr  setlfs
        jsr  open
        jsr  $a533
        lda  #$00
        sta  $9d
        jsr  $a659
        lda  #$0d
        jsr  chrout
        jmp  $a7ae
        nmi
        ldx  #$80
        ldy  #$00
wait    dey
        bne  wait
        dex
        bne  wait
back    lda  #$37
        sta  $01
        ldx  #$80
        ldy  #$07
        lda  #$00
labe    sta  $df00,y
        dey
        bpl  labe
        lda  #$8e
        sta  $033c
        lda  #$00
        sta  $033d
        lda  #$de
        sta  $033e
        lda  #$6c
        sta  $033f
        lda  #$fc
        sta  $0340
        lda  #$ff
        sta  $0341
        jmp  $033c
com     .byte 85,73
eind     =   *
na       =   eind+1
        .byte 0,0,0,0,0,0,0,0
        .byte 0,0,0,0,0,0,0,0
end     .end
```



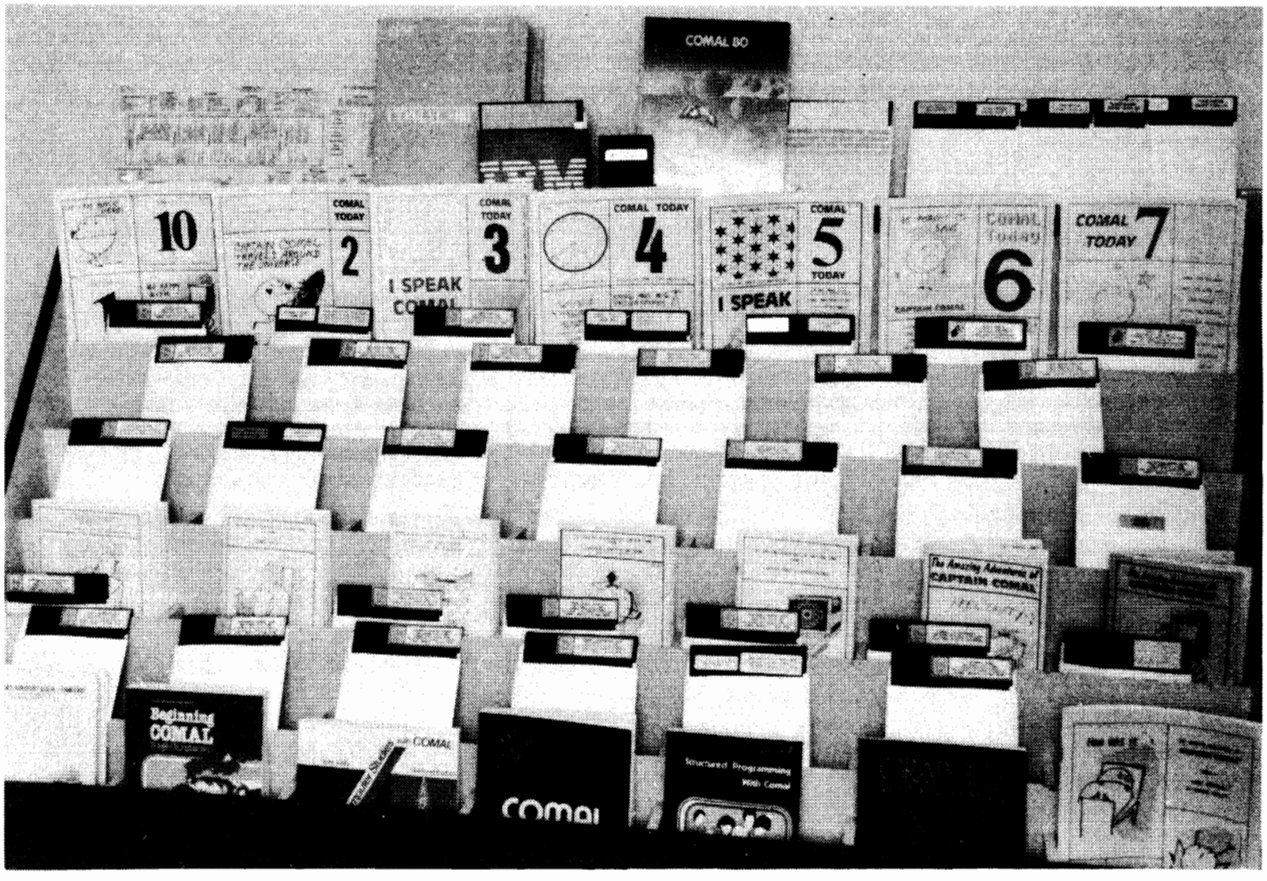
TODAY DISK 8
2.0 SIDE

```
hi
>---programs---<
beams
colorwheeldemo
concentration
create'bats
cross'reference
db'boot
db'define
db'help
db'labels
db'maintenance
db'menu
db'report
db'sort
db'squash
font'sprite
fun'print
illusion
interrupt/demo
make'data
make'err'english
make'font
morgage
run'basic'prog
scroll'message
seq'to'speed
soundex
speed'to'seq
sprite'converter
sprite-sample4
stack'space
star-zamara
statistics/demo
test'disk
test'pattern
trees
word'game
view'sprites
>--procedures--<
func.convert'bas
func.get'input
func.jdate
proc.alarm
proc.average
proc.cdate
proc.change'dev
proc.directory
proc.scroll'down
proc.siren
proc.stand'dev
proc.toggle'keys
proc.type
proc.window'down
proc.window'up
>--data-files--<
bat.demo
bat.loop
bat.sample
comal
dat.game'names
dat.information
db'data
db'help.def
db'help.lab
db'help.rpt
db'name
pkg.basic
pkg.dutch
shap.bat1
shap.bat2
shap.bat3
src.basic
```

TODAY DISK 8
0.14 SIDE

```
boot c64 comal
c64 comal 0.14
hi
menu
>---programs---<
art'nouveau
boot'dir'editor
colorwheel/demo
concentration
create'bats
depreciation
directory'editor
disk'edit/protct
display'seq'file
illusion
music'compiler
music'demo
seq'to'speed
sidmonitor
soundex
speed'to'seq
sprite'converter
sprite-sample4
trees
view'koala
view'sprites
>--procedures--<
comal'music.proc
loadshape.proc
saveshape.proc
sound.proc
use'sound.proc
>--data-files--<
-error-messages-
dance'hours.dat
dance'hours.sng
dir'editor.mem
dutch'errors
help.dat
music'player.mem
musicroutine.src
shap.'a'
shap.'c'
shap.'f'
shap.'k'
shap.'l'
shap.'m'
shap.'n'
shap.'o'
shap.'r'
shap.bat1
shap.bat2
shap.bat3
shap.harry00
shap.harry01
shap.harry02
shap.harry03
shap.harry04
shap.harry05
shap.harry06
shap.harry07
shap.harry08
shap.harry09
shap.harry10
shap.harry11
shap.queen
shap.santa0
shap.santa1
shap.santa2
```


IBM
IBM
COMAL-80
COMAL 80
COMAL 80
Commodore 64


COMAL 80
10
COMAL TODAY
2
COMAL TODAY
3
I SPEAK
COMAL
COMAL TODAY
4
COMAL
5
TODAY
I SPEAK
6
COMAL TODAY
7
Beginning
COMAL
comal
Structured Programming
With Comal

Price Protection Plan

As of June 17, 1985, every COMAL TODAY subscriber is protected automatically. If you buy any COMAL item from us - and the price goes down within 1 month - you automatically are entitled to a credit for the difference. Now you can buy a book or disk with confidence. We wonder how many other computer companies will be willing to follow our lead with this policy. Look at the CREDITS AVAILABLE listing below if you bought any COMAL items after July 1, 1985. Note: special prices offered at a show, conference, or similar event do not count as a price drop.

YOU ARE THE DEALER

As of August 1, 1985, all COMAL TODAY subscribers will automatically get a discount on nearly everything they buy from us. Most items list price allows for a dealer markup. Since most COMAL items are not sold through dealers, we decided to give our subscribers a dealer price. We will continue our special quantity prices to schools and groups as well.

THE NEW PRICE STRUCTURE

Keep it simple. What good is a discount if it takes you an hour to calculate? This is how our SUBSCRIBER DISCOUNT works:

$1 off each CHEATSHEET keyboard overlay
$2 off every Captain COMAL book
$3 off every other book
   plus only $2 for its matching disk
$4 off every Captain COMAL book/disk set
$5 off each COMAL 2.0 Deluxe Cartridge
All other disks are $9.75 each
$10 off each McPen Lightpen
$45 off IBM PC COMAL package

And due to our price protection plan, these prices are of course retroactive back to July 1, 1985. If you bought something on or after July 1 at a higher price, you can claim a credit.

NEW LOWER LIST PRICES

In addition to the discount, the list price on some COMAL items went down. Most notable is the Deluxe Cartridge Pak, now only $89.95 (down from $128.90). This is different from the EVERYTHING PAK which included one more book.

CREDITS AVAILABLE - PROTECTION PLAN

To claim your credit you must return your original invoice showing your purchase at the higher price. Circle the items and clearly mark the credit amount. The credit can be applied to any future order. Possible credits at this time are listed below:

CREDIT - ITEM	OLD	NEW
$2.00 CAPT COMAL GETS ORGNZD	12.95	10.95
$2.00 LIBRARY FUNC & PROC	12.95	10.95
$2.00 GRAPHICS PRIMER	12.95	10.95
$2.00 COMAL 2.0 PACKAGES	12.95	10.95
$2.00 COMAL FROM A TO Z	6.95	4.95
$2.00 COMAL WORKBOOK	6.95	4.95
$5.00 CART GRAPHICS & SOUND	9.95	4.95
$3.00 COMAL HANDBOOK	18.95	15.95
$5.00 BEGINNING COMAL	22.95	17.95
$3.00 FOUNDATIONS W COMAL	19.95	16.95
$3.00 STRUCTURED W COMAL	26.95	23.95
$2.95 Matching Disks	4.95	2.00
$1.00 Cheatsheet	3.95	2.95
10.00 McPen Light Pen	49.95	39.95
20.00 Disk Subscription	49.95	29.95
10.00 Starter Kit	29.95	19.95

Name:_____ (8/85)

Street:_____

City:_____ State:_____ ZIP:_____
NOTE: Pay by check, money order in US Dollars drawn on US Bank. Canada Postal US Dollar Money Orders are OK. VISA/MasterCard: PRINT card #, exp date below with your signature:

VISA / MC number:_____exp date:_____

Charge Orders: Signature:_____
>>> To qualify for subscriber prices we need your subscriber number: _____
>>> You automatically qualify if you start a new subscription now.

QNTY PRICE LIST/SUB PRICE - ITEM DESCRIPTION (all disks Commodore 1541 format)
(two disks may be supplied as a double sided disk)

SYSTEMS:
[ ] ____ **$89.95/$84.95** Deluxe Cartridge Package (2 books/5 disks/1 cartridge-$4 shipping)
[ ] ____ **$14.95/$10.95 COMAL QUICK 0.14 (fastloaded) with UTILITY DISK #2 and book**
[ ] ____ $29.95/$19.95 C64 COMAL 0.14 Starter Kit ($2 shipping)
[ ] ____ $11.00/$11.00 C64 COMAL 0.14 **$11 SPECIAL** AutoRun/Tutorial disk, COMAL FROM A TO Z
[ ] ____ $295/$250 IBM PC COMAL/IBM Denmark (Danish manual/free COMAL HANDBOOK-$2 shipping)

SUBSCRIPTIONS:
[ ] ____ **COMAL TODAY newsletter** >>> How many issues? _____ Start with #_____
($14.95 first 6 issues - $2 each added issue)-($2 for sample issue)-(Canada add $5)
[ ] ____ TODAY DISK subscription >>> How many? _____ Start with #_____
($49.95/$29.95 for first 6 disks - $5 each added disk)

BOOKS: (shipping $2 each)(matching disks are $19.95 or $2 to subscribers)
[ ] ____ $18.95/$15.95 COMAL HANDBOOK - optional disk [ ]
[ ] ____ $19.95/$16.95 FOUNDATIONS WITH COMAL - optional disk [ ]
[ ] ____ $26.95/$13.95 STRUCTURED PROGRAMMING WITH COMAL - optional disk [ ]
[ ] ____ **$20.95/$17.95** BEGINNING COMAL - optional disk [ ]
[ ] ____ $17.95/$14.95 COMMODORE 64 GRAPHICS WITH COMAL - disk not released yet
[ ] ____ $6.95/$4.95 COMAL FROM A TO Z (part of starter kit and $11 special)
[ ] ____ **$14.95/$10.95** CAPTAIN COMAL GETS ORGANIZED (with disk)
[ ] ____ $6.95/$4.95 COMAL WORKBOOK
[ ] ____ **$14.95/$10.95** COMAL LIBRARY OF FUNCTIONS & PROCEDURES (with disk)
[ ] ____ **$14.95/$10.95** GRAPHICS PRIMER (with disk)
[ ] ____ **$6.95/$4.95** CARTRIDGE GRAPHICS AND SOUND (part of Deluxe Cartridge Pak)
[ ] ____ **$14.95/$10.95** COMAL 2.0 PACKAGES (disk includes C64SYMB & MONITOR)
[ ] ____ $20/$15 CARTRIDGE TUTORIAL BINDER (with disk)(part of new Deluxe Cartridge Pak)

DISKS: (NOTE: all disks below are only $9.75 each to subscribers!).
[ ] ____ $94.05/$89.95 19 DISK SET (about 1000 programs for COMAL 0.14)
[ ] ____ $14.95/$9.75 BEST OF COMAL, (includes Disk Data Base)
[ ] ____ $14.95/$9.75 C64 COMAL SAMPLER, (part of Starter Kit)
[ ] ____ $7/$7 AUTO RUN DEMO DISK and TUTORIAL DISK (part of Starter Kit & $11 Special)
[ ] ____ $14.95/$9.75 Bricks TUTORIALS (2 sided BEGINNERS disk!)
[ ] ____ $14.95/$9.75 Modem Disk $14.95 (for COMAL 0.14 & 2.0).
[ ] ____ $14.95/$9.75 UTILITY DISK #1
[ ] ____ $14.95/$9.75 TODAY DISK **>>>> Which one(s):** _____
[ ] ____ $10/$9.75 USER GROUP DISK **>>>> Which one(s):** _____
[ ] ____ $14.95/$9.75 CARTRIDGE DEMO DISKS **>>>> Which one(s):** _____

OTHER
[ ] ____ OTHER: _____
[ ] ____ $3.95/$2.95 KEYBOARD OVERLAY for C64 COMAL 0.14 - CHEATSHEET (add $1 handling)
[ ] ____ $49.95/$39.95 McPen Light Pen (with COMAL 2.0 Demo Disk)($2 shipping)
[ ] ____ **$9.95/$5.95 Royal Blue COMALite Shirt** (circle size: ADULT: S / M / L / XL)
=====

TOTAL_____ + _____ Shipping (min $2) = TOTAL PAID $_____ (WI add 5% sales tax)
Mail To: COMAL Users Group USA, 6041 Monona Drive, Madison, WI 53716 or call 608-222-4432

Yes, I want COMAL TODAY. Give me _____ issues.
($14.95 for 1st 6, $2 each after that)

This is a: [ ] New Subscription
[ ] Renewal
My subscriber # is:_____

Name:_____

Address:_____

City:_____State:_____ZIP:_____

[ ] Check enclosed [ ] Visa [ ] MasterCard

Card # _____expires_____

Signature:_____

From: _____

_____

_____

PLACE
STAMP
HERE

TO: COMAL USERS GROUP, USA, LIMITED
5501 GROVELAND TERRACE
MADISON, WI 53716-3251

Subscriber's Specials

Here are this issues specials for subscribers only. If you subscribe now, you can take advantage of these specials at the same time. If you are a current subscriber you **MUST** include your subscriber number with your order, or we will not honor the special prices. Your subscriber number is printed on the newsletter mailing label. New subscribers get these prices automatically and a number will be assigned with the order.

EVERYTHING

As a subscriber you now get a special price on virtually everything. The subscriber specials continue in addition to the regular discount.

BUY ONE DISK - GET ONE FREE

Stock up on all the COMAL disks now. For every disk you buy, pick one to get for FREE. This excludes the matching book disks and the $7 special set. The photo below shows some of our variety of COMAL disks.

DISK SUBSCRIPTION ONLY $29.95

We hate to have our readers type in the programs in the newsletter. Right from our first issue of COMAL TODAY a matching disk has been available. Now it is affordable as well. We could not find a lower priced disk subscription offer anywhere. And our disks are full of programs on BOTH sides. That is better than half price!

THE SHADOW KNOWS

99.9% EFFECTIVE

Shadow is a new and revolutionary way to duplicate even your most protected software. It encompasses all the latest advances in software, as well as a highly sophisticated piece of hardware. This is absolutely **the best** utility available today. "It will even copy the other copy programs." Because of Shadow's unique abilities, we feel DOS protection is a thing of the past.

By the time you place your order we expect the Shadow to copy 100% — that's right, 100% — of **all** software available for the C-64.

Order by phone 24 hrs./7 days or send cashier's check/money order payable to Megasoft. Visa, MasterCard include card # and exp. date. Add $3.50 shipping/handling for continental U.S., $5.50 for UPS air. CODs add $7.50. Canada add $10.00 Other foreign orders add $15.00 and remit certified U.S. funds only. Distributors invited and supported.

MegaSoft LTD.

INTRODUCTORY OFFER **$89.95**

P.O. Box 1080 Battle Ground, Washington 98604
Phone (206) 687-5116 • BBS 687-5205 After-Hours Computer-to-Computer Ordering